Efficient Time-Interrupted and Time-Continuous 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, Taiwan 106, R.O.C.

This paper presents efficient time-interrupted and time-continuous collision detection procedures for polyhedral objects which consist of convex polygons and perform arbitrary translating and/or rotating motions in a 3-D graphical environment. The time-interrupted collision detection procedure can find the exact position of a collision event in a time step while the time-continuous procedure can determine both the approximate time and position of a collision event. Each procedure first localizes the object-to-object collision events using a "space cell" method, which divides the 3-D environment into small cells in order to reduce the number of object pairs required for collision detection. An "azimuth-elevation map" method is then proposed which rapidly selects the polygons within or crossing the overlap region between two possibly colliding objects by presorting their vertices in the spherical coordinates system. Subsequently, a divide-and-conquer method that takes advantage of a recursive scheme is devised to moderate the number of polygons which need to be checked by means of a polygon-to-polygon intersection test. To deal with time-interrupted and time-continuous collision detection, two respective polygon-to-polygon intersection testing methods based on a hierarchical scheme are developed to diminish the number of unnecessary computations. All the experiments were performed using computer simulation. So far, the experimental results from our proposed methods are very encouraging.

Keywords: collision detection, polyhedral object, bounding volume, space cell, azimuth-elevation map

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 used in virtual reality to achieve interaction between users and virtual worlds. To quickly respond to input from users, efficient collision detection among polyhedral objects (called objects for short) must be accomplished in a 3-D graphical environment. The schemes for collision detection are categorized into two major approaches. We call them "time-interrupted" and "time-continuous" collision detection. The first approach detects collision events in each time step to determine the status of objects in a 3-D environment and then makes associative responses. Because detection is executed after a time interval, some collision events might be lost and the time when such a collision event has occurred is not known. This is not acceptable for most of simulation applications. The second approach can predict the collision time by...
tracing the velocities and/or acceleration of objects on certain assumptions. However, the computation load is quite heavy, so this method is often used in applications without regard to real-time responses.

Objects in a 3-D environment are frequently represented by polygons or parametric surfaces [1]. Dividing the space into smaller subspaces occupied by objects individually can reduce the number of polygon pairs to be compared [2-4]. For complicated 3-D objects, hierarchical representation schemes can be adopted to speed up the collision detection [4-7]. Additionally, bounding volume schemes also save the computing time of collision detection and are extensively used in many applications [8-10]. In general, each of time-interrupted and time-continuous collision detection approaches can be roughly divided into two phases: an object-to-object overlap test and a polygon-to-polygon intersection test. The difference between time-interrupted and time-continuous collision detection is that the latter must predict the collision time and position, but the former only determines whether a collision event happens or not. 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 which may have collided. To reduce the computation load, many methods have been proposed in the literature [3, 8, 9, 11].

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 so as to entirely surround the object. In 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 the pairs of objects in the environment [8, 9]. In fact, only a few pairs of objects need to be detected. Accordingly, we can just trace the moving objects in the environment in order to reduce the computation load. Before the polygon-to-polygon intersection test is performed, the polygons within or crossing the overlap region of two possibly colliding objects must be found. The most common way is to test all the polygons of the two objects against the overlap region. This increases the computation load. Another improved method divides the bounding box of an object into smaller cells, each of which stores the associated constituting polygons [8]. While polygons within or crossing the overlap region are being found, 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 this will require multiple tests.

Spatial occupancy enumeration strategies, including octrees [5, 12], binary space partitioning (BSP) trees [13], and the successive spherical approximation (SSA) representation [2], are also used to check the bounding volume overlap and find polygons within or crossing the overlap region. Their common feature is that the space occupied by an object is decomposed into many subspaces in the 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 rebuilt after the object completes geometrical transformations (translations and/or rotations). Again, as mentioned above, the size of the smallest unit of the structure can not be well determined, and a polygon may span several subspaces [9]. This will increase the computation load of the polygon-to-polygon intersection test. Furthermore, the aforementioned strategies will enlarge the amount of memory needed to store the hierarchical structures. A more detailed description of this approach will be given in a later section.
In order to predict the collision time of two objects, some methods use bounding spheres to encircle the objects. Given the velocities of the centers of the two bounding spheres, the collision time can be derived from some algebraic manipulations when the distance between the centers is equal to the summation of the radii of the spheres. The swept volume method, which examines the intersection of the volumes swept by the bounding volumes or polygons over the path on the moving, is also widely used to acquire the approximate collision time [6, 7, 14]. When the path is changed, the swept volume must be regenerated, and this will increase the computation load in interactive applications. The collision time is calculated by testing the intersection of the edges and faces of the swept volumes. However, when objects perform rotations, the swept volumes may contain curved edges or twisted faces. These situations will increase the difficulty of finding the intersection position.

2. RELATED WORK AND OUR PROPOSED METHODS

The main purpose of collision detection is to find the interference between two objects in a 3-D graphical environment and to make associative responses. There are two principal approaches to representing objects, e.g., spatial occupancy enumeration and bounding volumes, to perform object-to-object overlap and polygon-to-polygon intersection tests. An object-to-object overlap test is used to check whether a collision event occurs between two objects whose polygons are embraced by bounding volumes, such as boxes and spheres. A polygon-to-polygon intersection test is employed to ensure which two polygons will interfere with each other and to find the position where the intersection will occur. The following subsections will survey some related works on collision detection.

2.1 The Spatial Occupancy Enumeration Approach

In this subsection, we will describe three spatial occupancy enumeration strategies: octrees, BSP trees, and the SSA representation, which will be compared with the bounding volume approach and our proposed methods.

2.1.1 Octrees

The octree configuration, a way of representing the space occupied by an object, adopts a recursive cubic decomposition of the universe space [12]. There are three kinds of nodes in an octree: black, gray, and white nodes. A black node means the corresponding subspace completely contains parts of an object; otherwise, a gray node is used to represent a subspace which partially contains an intact object. If the subspace contains no part of an object, it is represented by a white node. Collision detection can be accomplished by traversing the octrees of different objects in parallel [3, 10].

The mean computational complexity of detecting an overlap event among \( n \) objects is \( O(Kn) \), where the number of nodes of the tree is \( K \) on average [3]. Although its complexity is better than that of the bounding volume approach, \( O(n^2) \), it requires more memory storage [10]. Another problem is that octrees must be regenerated in each time step when the associated object performs geometrical transformations.
When the overlapping nodes of two octrees are found, the polygon-to-polygon intersection test is then executed. The polygons of the different objects with the same overlapping node form the polygon pairs, and the intersection test is subsequently carried out. An advanced method first builds an octree of one object’s polygons within or crossing the overlap region [9]; afterward, the other objects’ polygons within or crossing the same overlap region are inserted. By traversing the octree, the minimum polygon pairs can be obtained.

2.1.2 BSP trees

Constructing a BSP tree is another method used to detect collision events among objects. A BSP tree represents a recursive subdivision or hierarchical partitioning of an n-dimensional space by some convex subspaces [13]. The construction of a BSP tree is a recursive process which takes a subspace and partitions it using any hyperplane that intersects the interior of that subspace. An octree can be alternately represented by a BSP tree. When the polygons of an object act as hyperplanes, the BSP representation can be employed to group the polygons of the object, so it is not necessary to test the bounding volumes of objects. The collision detection procedure with a BSP tree is achieved by inserting the polygons of the other objects into the BSP tree to find out potentially colliding polygon pairs.

2.1.3 The SSA representation

The SSA representation is one of the approaches which adopt spheres as the bounding volumes of objects, and is made up of a hierarchy of representation levels based on dividing the encircling sphere into spherical sectors [2]. This representation consists of the following levels:

- Level 0: object bounding spheres;
- Level 1: facial bounding spheres;
- Level 2: approximate subdivision of a face;
- Level 3: faces.

According to the spherical coordinates of the center of the bounding sphere, the sectors in the overlap region can be easily obtained, and the levels described above are appropriately used to eliminate unnecessary computations.

2.2 The Bounding Volume Approach

Bounding volumes are applied to save computing time used in comparing all the polygons of an object. The common shapes of bounding volumes are boxes and spheres. The corresponding bounding volumes must be updated when objects perform geometrical transformations. While using bounding spheres, the task can be simplified to update the coordinates of their centers only. When using bounding boxes, it is a time-consuming task to compare all the coordinates of the vertices of objects to obtain the updated bounding boxes. An alternative method is to store the original bounding box as a local one [8]. When an object performs geometrical transformations, the associated local bounding box also
updates its vertices in step with the transformations. Therefore, all the updated vertices are compared to get the global bounding box whose edges are aligned with the axes of the coordinates system. The global bounding box can be subsequently used for the bounding volume overlap test. However, the global bounding box is often larger than the local one. This leads to unnecessary computations in the polygon-to-polygon intersection test. A pair of global bounding boxes has an overlap region if and only if their intervals intersect each other in all three dimensions. The following is the general way of the bounding box approach used to detect collision events:

1) For each object, generate its own bounding box.
2) Perform the overlap test among all the objects.
3) If an overlap event occurs, find those polygons intersecting the overlap region.
4) Test the polygon pair (each polygon of the pair belongs to different objects) to find the intersection.
5) Update the bounding boxes if the objects execute geometrical transformations; then go to Step 2.

The computational complexity of the bounding box overlap test is $O(n^2)$, where $n$ is the number of objects. A method for accelerating the test is described in [11]. The idea behind this method is to sort the extents in each dimension and to then search those extents in one dimension each time. If an overlap event is found, the current extent is recorded and the next one tested. Only those extents which mutually overlap in a dimension are further checked. The computational complexity can be reduced to $O(n \log n)$, which neglects the time spent in a linear search among the lists of the extents.

For the polygon-to-polygon intersection test, one approach is to divide the bounding volume into cells [8], each of which keeps a list of the constituting polygons. The cells in the overlap region are tested with the cells of the other objects in the same region. Another approach first builds an octree of the polygons of one object completely or partially located in the overlap region [9]. After that, the polygons of the other object are inserted into the octree to find potentially colliding polygon pairs in each black node and to then perform the polygon-to-polygon intersection test.

2.3 Our Methods

The spatial occupancy enumeration approach described earlier can moderate the number of polygon pairs employed in the test; however, this increases the memory storage requirement [10] and needs more update time than the bounding volume approach does. Another problem is that when a polygon spans different subspaces, duplicate tests of the same polygon will result. The solution is to either identify a number of polygons to avoid duplicate tests [3], or to split the polygon into pieces associated with the spanned subspaces [13]. Both approaches demand extra computing resources.

To accelerate the object-to-object overlap test, neither the spatial occupancy enumeration nor the bounding volume approach is adopted. Our strategy is to divide the 3-D cyber space into equal cells, each of which is called a “space cell.” When an object executes geometrical transformations, the cells containing parts of the object will be made a sign. After all the objects finish their transformations, only those cells with the sign will be
checked. Therefore, the number of object pairs which need to be checked is considerably diminished. In consequence, this space cell method is more efficient than the exhaustive method based on the spatial occupancy enumeration or the bounding volume approach, even when the worst case, where all the objects are located in a very few cells, occurs because the update time required for the former is much less than that for the latter.

Since a point located in the spherical coordinates system retains 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 crossing the overlap region can be easily selected using our “azimuth-elevation (A-E) map” method. As opposed to employing the update time of the data structure of spatial occupancy enumeration, this method spends little time in managing the A-E map by transforming the local spherical coordinates of an object into the global Cartesian coordinates of possibly colliding objects. The potentially colliding polygons can be directly selected from the A-E map of the object without testing all the polygons of the possibly colliding objects against the overlap region.

No matter what results are obtained using the methods mentioned above, the initially 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 which moderates the number of polygon pairs needed for testing. This procedure repeatedly compares the bounding boxes in the two updated lists of polygons within or crossing the overlap region of the two objects until the size and the location of the overlap region are unchanged. Such a method will not generate the data structure like spatial occupancy enumeration and will avoid the “span problem” of polygons. As for the polygon-to-polygon intersection test, we incorporate a hierarchical scheme [9] and extend the Cyrus-Beck algorithm [15]. The whole test procedure includes three steps: an overlap test on polygons embraced with bounding boxes, a crossing test on an edge and a polygon, and an inside test on an intersection point and a convex polygon. After executing these steps, the crossing position of the two polygons can be obtained if they intersect.

Time-continuous collision detection is explored in simplified cases. Our proposed method is excluded in testing the intersection of the swept volumes of objects or polygons. The constituents of an object, vertices, edges, and polygons, are employed to determine the earliest time when two objects will collide with each other. Two collision conditions are considered to test: one is the “edge-to-edge” case and another is the “vertex-to-polygon” case. Then the collision position is acquired. By using the two tests, the problems caused by curved edges and twisted faces in the swept volume approach can be avoided.

3. TIME-INTERRUPTED COLLISION DETECTION

The time-interrupted collision detection procedure which we will present in this section can cope with translating and/or rotating objects in a 3-D environment. In this procedure, the processing time is divided into a sequence of unit steps, and the geometrical position of each object is updated in each time step. The collision detection procedure consists of three stages. First, we try to reduce the testing times between all the objects surrounded by bounding volumes using space cells. In this manner, the collision event can be located by
only paying attention to the cells which the objects enter or exit. Second, we take advantage of the orientation invariance characteristic of spheres to find the constituting polygons of an object which may collide with those of another object. This will save time in choosing certain polygons of the two possibly colliding objects, without comparing all their polygons against the overlap region. Third, we propose a divide-and-conquer scheme to speed up the polygon-to-polygon intersection test. The objects we consider are vertices which further constitute edges and polygons. All these polygons are convex ones whose vertices are stored in a counterclockwise sequence. In this way, the normal vector of a polygon can be easily calculated.

### 3.1 The Object-to-Object Overlap Test Based on Space Cells

This subsection describes the first stage of our time-interrupted 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 it is easy to check for overlap (if the distance between the centers of two spheres is less than the summation of the radii of the two spheres). When objects are rotated or translated, all the bounding spheres merely update their center coordinates. And not any update operations are required for rotated objects.

Because only moving objects have opportunities to collide with other objects, we can focus on the rotating and/or translating objects in each time step. In addition, only the objects near the moving ones need to be checked, not all the objects in the virtual world. A data structure, which we call a space cell, is used to record the status of objects in the world. To begin with, the world is partitioned into a set of cubic cells. Then, each cell can accept or cancel the registrations of objects when they move into or off the region of the cell. The following are the steps in our method:

1) If an object has 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, compute the serial numbers of the cells which 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 out whether an overlap event has occurred or not. Otherwise, no overlap event exists.
4) If an overlap event happens, pass the two possibly colliding objects to the second stage to determine the pair of polygons that intersect.

With this method, every object keeps the data structure as 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 \((x_{\text{center}}, y_{\text{center}}, z_{\text{center}})\) and radius \(r\) as follows:

\[
\begin{align*}
  p_1 &= (x_1, y_1, z_1) = (x_{\text{center}} - r, y_{\text{center}} - r, z_{\text{center}} - r), \\
  p_2 &= (x_2, y_2, z_2) = (x_{\text{center}} - r, y_{\text{center}} - r, z_{\text{center}} + r), \\
  p_3 &= (x_3, y_3, z_3) = (x_{\text{center}} - r, y_{\text{center}} + r, z_{\text{center}} - r), \\
  p_4 &= (x_4, y_4, z_4) = (x_{\text{center}} - r, y_{\text{center}} + r, z_{\text{center}} + r), \\
  p_5 &= (x_5, y_5, z_5) = (x_{\text{center}} + r, y_{\text{center}} - r, z_{\text{center}} - r), \\
  p_6 &= (x_6, y_6, z_6) = (x_{\text{center}} + r, y_{\text{center}} - r, z_{\text{center}} + r), \\
  p_7 &= (x_7, y_7, z_7) = (x_{\text{center}} + r, y_{\text{center}} + r, z_{\text{center}} - r), \\
  p_8 &= (x_8, y_8, z_8) = (x_{\text{center}} + r, y_{\text{center}} + r, z_{\text{center}} + r).
\end{align*}
\]
The eight vertices constitute the 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:

\[
N_i = I_z_i \cdot n_x \cdot n_y + I_y_i \cdot n_x + I_x_i
\]

with

\[
I_x_i = \frac{x_i - x_{\text{min}}}{2r}, \quad I_y_i = \frac{y_i - y_{\text{min}}}{2r}, \quad \text{and} \quad I_z_i = \frac{z_i - z_{\text{min}}}{2r}
\]

for \( i = 1, 2, ..., 8 \),

where \( x_{\text{min}} \), \( y_{\text{min}} \), and \( z_{\text{min}} \) are the coordinates of the minimum extent of the world; \( n_x \) and \( n_y \) are the numbers of partitions in the first and second dimensions, respectively; \( r \) is the radius of the moving bounding sphere. It is noted that we take the nearest integers of \( I_x_i \), \( I_y_i \), and \( I_z_i \) with the rounding-off method.

![Fig. 1. The eight vertices of the cube surrounding a bounding sphere.](image)

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 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. In 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. Fig. 2 illustrates the partitioning 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.

The data structure of the space cell is defined as:

```cpp
struct SpaceCell {
    bool NeedCheck;
    bool DynamicObject[N];
    bool StaticObject[M];
};
```
The Boolean flag NeedCheck is used to identify if the cell has been detected in Step 3 of the method mentioned above. There might be more than one moving object registered by the same cell, which will cause the cell to be accessed several times in Step 3. The flag is necessary to avoid extra computations. Assume that there are currently \( N \) moving objects and \( M \) resting objects in the visible scope of the virtual world. Two Boolean arrays are employed to indicate whether the cells are empty or not. When the \( i \)-th moving object is registered, the \( i \)-th element of the array DynamicObject is TRUE; otherwise, the corresponding element is FALSE when the cell cancels the registration. In the same manner, resting objects are also managed using the array StaticObject. By distinguishing moving objects from resting ones, some computing time can be saved because the resting objects have no opportunities to collide with the other resting ones.

Another problem is perhaps encountered in Step 3. That is, two possibly colliding objects might be simultaneously registered by several cells, so these cells will be accessed in Step 3, after which the two objects will be passed to Step 4 more than one time. To prevent this situation from happening, a 2-D Boolean array is adopted to express if the pair of objects has been checked. This array is always cleared in Step 1.

### 3.2 Selecting the Polygons Within or Crossing an Overlap Region

In the second stage, the pairs of possibly colliding objects obtained in the first stage are checked where the collision events occur. The commonly used method often tests all the constituting \( m_1 \) and \( m_2 \) polygons in the pair of objects to filter out those polygons which cross or are within the overlap region, and further tests them in pairs. Essentially, the computational complexity is \( O(m_1 m_2) \) and can not be reduced. In this subsection, we will propose a method to save the computing time used to test all the polygons which cross or are within the overlap region. In order to reduce 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 which share the same vertex. Furthermore, we need an array in which some pieces of sorted information are stored.

#### 3.2.1 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 \((x_{\text{center}}, y_{\text{center}}, z_{\text{center}})\) of an object are derived from taking an average of the

![Fig. 2. An example of partitioning a space with two objects.](image)
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 \((\rho, \theta, \phi)\) relative to the center coordinates as:

\[
\begin{align*}
\rho &= \sqrt{x'^2 + y'^2 + z'^2}, \\
\theta &= \tan^{-1} \frac{y'}{x'}, \\
\phi &= \tan^{-1} \frac{\sqrt{x'^2 + y'^2}}{z'}
\end{align*}
\]

where \(0^\circ \leq \theta < 360^\circ\), \(0^\circ \leq \phi < 180^\circ\), \(x' = x - x_{\text{center}}\), \(y' = y - y_{\text{center}}\), and \(z' = z - z_{\text{center}}\).

After this transformation, the maximum value of \(\rho\) is selected as the radius of the bounding sphere of the object. Subsequently, all the vertices are sorted according to the values of the azimuth and elevation angles, \(\theta\) and \(\phi\), respectively, and placed in a 2-D array, called an A-E map. Fig. 3 shows an A-E map, each element of which keeps the spherical coordinates of a vertex.

![Fig. 3. An azimuth-elevation map in the form of a 2-D array.](image)

When a collision event between two objects may happen, the following method is applied to find their constituting polygons that may collide with each other. Denote the two objects as \(\text{Object 1}\) and \(\text{Object 2}\).

1. Transform the center coordinates of \(\text{Object 2}\) into spherical coordinates referring to that of \(\text{Object 1}\); consequently, the direction of \(\text{Object 2}\) relative to \(\text{Object 1}\) is known.
2. From the sphere boundary, the crossing angles \(\psi_{1\theta}\) and \(\psi_{1\phi}\) of \(\text{Object 1}\) are calculated as follows:

   If \(\delta \geq r_2\), 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 the polygons),

where $r_1$ and $r_2$ are the radii of the bounding spheres of Object 1 and Object 2, respectively, and $\delta$ is the distance between the centers of the bounding spheres. Fig. 4 shows an illustrative example for the case of the first if-statement.

![Fig. 4. Illustration of two overlapping bounding spheres.](image)

(3) Select those vertices on the A-E map of Object 1 whose azimuth and elevation angles satisfy $\theta_{\text{Object2}} - \psi_{1\theta} \leq \theta \leq \theta_{\text{Object2}} + \psi_{1\theta}$ and $\phi_{\text{Object2}} - \psi_{1\phi} \leq \phi \leq \phi_{\text{Object2}} + \psi_{1\phi}$, as Fig. 5 illustrates. If $\rho$ is less than $\eta$, then the associated vertex is discarded.

![Fig. 5. An example of a mapped overlap region between Object 1 and Object 2 on the A-E map.](image)

(4) List those polygons containing the selected vertices.
(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 to find the collision position.
Here, a special case needs to be discussed. As shown in Fig. 6, the mapped overlap region extends 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 equations of the first if-statement in Step 2 can solve this problem by enlarging the region to include all possible vertices, but such a strategy also increases the number of polygons managed in Step 6. If the object does not have big polygons, the following manipulation is preferred:

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

(3)

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

3.2.2 The solution to rotated objects

The method described in the previous subsection can only deal with the situation in which 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 in order to select the polygons which may collide with those of another object. The way to solve this problem is that for each
object, we stored its unit vectors of three dimensions since the object was created. When an object performs rotations, its unit vectors are also rotated in order to acquire the local axes of the coordinates of the object. For a rotated object, the transformation of the global Cartesian coordinates into the local spherical ones is accomplished as follows. First, the global Cartesian coordinates of the center of a potentially colliding object are transformed into the local ones of the associated bounding sphere of the rotated object by projecting the global coordinates into the constituting axes of the three rotated unit vectors, and these local Cartesian coordinates are then transformed into the spherical ones of the rotated object. Therefore, the correct position on the A-E map of the potentially colliding object is received. In this way, the computing time needed to resort every vertex on the A-E map is reduced for the rotated objects.

3.3 Speeding up the Polygon-to-Polygon Intersection Test

Previously, we obtained two lists of polygons, list 1 and list 2, from two different objects possessing an overlap region. These two lists can include \( m_1 \) and \( m_2 \) polygon pairs, where \( m_1 \) is the number of polygons in list 1 and \( m_2 \) is the number of polygons in list 2. In order to detect the occurrence of collision events, we must test each polygon pair for interference. When we test in such an exhaustive manner, the computational complexity is \( O(m_1 m_2) \). The following is our proposed method, which reduces the computing time.

3.3.1 Overlap test on polygons embraced with bounding boxes

We adopt the bounding box as the auxiliary data structure to simplify the polygon-to-polygon intersection test. In order to save the computing cost, we must choose polygon pairs which seem to be interfered, not all the polygon pairs. To achieve this goal, we employ the “mini-max test” mentioned in a hidden-surface removal algorithm [16]. For each polygon, a data structure keeps both the maximum and minimum of the three coordinates of the vertices in its bounding box. If the mini-max test result is positive, then the polygons do not interfere with each other. But, if the bounding boxes do overlap, we have still not confirmed whether the polygons within them experience interference. In such a case, we must carry out a further test.

3.3.2 Crossing test on an edge and a polygon

A hierarchical test as described in [9] is used to check for interference events 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), then there is no edge crossing the plane, and there are no intersections. Otherwise, if the vertices of Poly1 lie on different sides of the plane formed by Poly2 and vice versa, then an interference event has been 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 with the same sign indicate vertices on the same side of the plane. If the distances have different signs, then the vertices of the polygon lie on different sides of the plane. If interference events exist, then we will perform the following test to confirm whether the two polygons really intersect each other.
3.3.3 Inside test on an intersection point and a convex polygon

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

$$p = p_i + |d_i|(p_j - p_i) / (|d_i| + |d_j|).$$  \hspace{1cm} (4)

Subsequently, the 2-D Cyrus-Beck algorithm [15] is extended to the 3-D case to test whether the intersection point is inside a convex polygon. Because all the vertices are arranged counterclockwise to determine the visibility of a polygon, the outward normal vector of the edge $p_ip_j$ is calculated by $N_e = e_{ij} \times N$, referring to Fig. 9, where $e_{ij}$ is the vector of the edge $p_ip_j$ and $N$ 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 the edges of the polygon from the above calculations, then the point $p$ is inside the polygon; if not, it is outside (see Fig. 10). The crossing line of the two interferring polygons can be found by connecting the intersection points of the two polygons.

Subsequently, the 2-D Cyrus-Beck algorithm [15] is extended to the 3-D case to test whether the intersection point is inside a convex polygon. Because all the vertices are arranged counterclockwise to determine the visibility of a polygon, the outward normal vector of the edge $p_ip_j$ is calculated by $N_e = e_{ij} \times N$, referring to Fig. 9, where $e_{ij}$ is the vector of the edge $p_ip_j$ and $N$ 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 the edges of the polygon from the above calculations, then the point $p$ is inside the polygon; if not, it is outside (see Fig. 10). The crossing line of the two interferring polygons can be found by connecting the intersection points of the two polygons.
3.3.4 A divide-and-conquer procedure

A divide-and-conquer procedure is employed to save the computing time for detecting collision events among objects. To begin with, the bounding boxes of the two lists of polygons obtained using the A-E map method are generated. Then the overlap test using bounding boxes is applied to check for overlap events. If no overlap occurs, then the procedure is terminated; otherwise, the mini-max test method is adopted to pick those polygons within or crossing the overlap region in order to compose the other two lists of polygons. In the same manner, two newly generated lists are used to repeatedly perform the above procedure until the overlap region does not change. Fig. 11 shows the pseudocode of this procedure in a recursive scheme.

**Interference Test (the overlap region R, list1, list2)**

```
{ 
    generate the bounding box B1 of list1;  
    generate the bounding box B2 of list2;  
    if (the overlap region R1 between B1 and B2 exists) {  
        if (R = R1)  
            perform the polygon-to-polygon intersection test for all the polygons within or crossing the overlap region R;  
        else {  
            get the polygon list sub-list1 from the overlap region R1 and list1;  
            get the polygon list sub-list2 from the overlap region R1 and list2;  
            Interference Test (R1, sub-list1, sub-list2);  
        }  
    }  
} 
```

Fig. 11. The divide-and-conquer procedure for checking for interference between two polygons.
4. TIME-CONTINUOUS COLLISION DETECTION

In the previous section, collision events were detected once every a regular time. When the movement of objects is too large, some collision events might be lost during a time step. In order to avoid this situation, some algorithms have been developed to decide the time when a collision event happens.

A four-dimensional structure method used is to compute the volumes swept by the objects over their motions [14]. If these swept volumes intersect, then a collision event is detected. In this way, the collision time can be predicted, and associative computations are not necessary until the time of collision nears. However, if the velocity and/or the acceleration of a movable object are changed by external inputs, the structure must be generated again. An algorithm based on the scheme of space-time bounds was proposed to solve this problem [6, 7]. The general formulation of the collision time of a moving point and a moving polygon was described in [15], but not for two edges moving toward each other. In common practice, the movement paths of objects are assumed to be linear, and the velocities of the constituting vertices are fixed. This may result in some errors when polygons are rotated, as illustrated in Fig. 12.

In the following subsections, we will propose the solutions used to determine the collision time and position of a point moving close to a moving polygon as well as to determine those of two edges moving close to each other. To reduce the computation load, the assumptions of linear motion and fixed velocity mentioned above are adopted in calculating the approximate values of the collision time and position. Accordingly, the time-continuous collision detection procedure is accomplished.

4.1 Determining the Collision Time of Two Moving Objects

After the object-to-object overlap test as described in subsection 3.1, collision detection can further be resorted to a polygon-to-polygon intersection test, which is performed using a vertex-to-polygon collision test and an edge-to-edge collision test. In the instant of occurring in a collision event, the location of collision must be a point. It is difficult to determine the time at which two moving points will pass through the same location. A
method described in [17] models the problem as follows: suppose there are two positions, \( p_a \) and \( p_b \), which are penetrated by two edges or polygons at different moments. Fig. 13 demonstrates that one edge or polygon in Object 1 passes through \( p_a \) and \( p_b \) at time \( t_a \) and \( t'_a \), whereas another edge or polygon in Object 2 passes through \( p_b \) and \( p_a \) at time \( t_b \) and \( t'_b \), respectively. Then, the collision time \( t_c \) is calculated by

\[
t_c = \frac{t'_b - t_a}{(t'_b - t_a) + (t'_a - t_b)}.
\]

\[ (5) \]

Fig. 13. Illustration of the elapsed time and varied position for two edges or polygons moving between \( p_a \) and \( p_b \).

4.2 Dynamic Edge-to-Edge Collision Test

Assume that edge \( E_1 \) moves at the fixed velocities \( v_1 \) and \( v_2 \) for vertices \( p_1 \) and \( p_2 \), respectively, while \( E_2 \) moves at the fixed velocities \( v_3 \) and \( v_4 \) for \( p_3 \) and \( p_4 \), respectively, as shown in Fig. 14. Using the 3-D components of these parameters, we can write two symmetric equations for lines \( L_1 \) and \( L_2 \) respectively containing edges \( E_1 \) and \( E_2 \) in the same form:

\[
\begin{align*}
\frac{x - (p_{1x} + v_{1x}t)}{p_{2x} - p_{1x} + (v_{2x} - v_{1x})t} &= \frac{y - (p_{1y} + v_{1y}t)}{p_{2y} - p_{1y} + (v_{2y} - v_{1y})t} = \frac{z - (p_{1z} + v_{1z}t)}{p_{2z} - p_{1z} + (v_{2z} - v_{1z})t} \\
\end{align*}
\]

\[ (6a) \]

Fig. 14. Two moving edges approaching each other.
Then, the parametric equations of $x = p_{1x} + v_{1x}t$ and $z = p_{1z} + v_{1z}t$ are:

\[
\frac{x - (p_{1x} + v_{1x}t)}{p_{2x} - p_{1x}} = \frac{y - (p_{1y} + v_{1y}t)}{p_{2y} - p_{1y}} = \frac{z - (p_{1z} + v_{1z}t)}{p_{2z} - p_{1z}}. \tag{6b}
\]

What follows is the method used to find the intersection point of $L_1$ and $L_2$. First, we let Eq.(6a) equal a constant $s$:

\[
\frac{x - (p_{1x} + v_{1x}t)}{p_{2x} - p_{1x} + (v_{2x} - v_{1x})t} = \frac{y - (p_{1y} + v_{1y}t)}{p_{2y} - p_{1y} + (v_{2y} - v_{1y})t} = \frac{z - (p_{1z} + v_{1z}t)}{p_{2z} - p_{1z} + (v_{2z} - v_{1z})t} = s. \tag{7}
\]

Then, the parametric equations of $L_1$ are:

\[
\begin{align*}
x &= p_{1x} + v_{1x}t + s(p_{2x} - p_{1x} + (v_{2x} - v_{1x})t), \\
y &= p_{1y} + v_{1y}t + s(p_{2y} - p_{1y} + (v_{2y} - v_{1y})t), \\
z &= p_{1z} + v_{1z}t + s(p_{2z} - p_{1z} + (v_{2z} - v_{1z})t). \\
\end{align*}
\tag{8a,8b,8c}
\]

Let $t$ in Eq.(6b) equal zero. Therefore, the equation becomes

\[
\frac{x - p_{3x}}{p_{4x} - p_{3x}} = \frac{y - p_{3y}}{p_{4y} - p_{3y}} = \frac{z - p_{3z}}{p_{4z} - p_{3z}}. \tag{9}
\]

and we have the following three equations:

\[
\begin{align*}
(x - p_{3x})(p_{4x} - p_{3x}) &= (y - p_{3y})(p_{4y} - p_{3y}), & \tag{10a} \\
(y - p_{3y})(p_{4z} - p_{3z}) &= (z - p_{3z})(p_{4x} - p_{3x}), & \tag{10b} \\
(z - p_{3z})(p_{4y} - p_{3y}) &= (x - p_{3x})(p_{4z} - p_{3z}). & \tag{10c} \\
\end{align*}
\]

By substituting Eqs.(8a), (8b), and (8c) into Eqs.(10a), (10b), and (10c), we can obtain the following equations:

\[
\begin{align*}
a \cdot st + b \cdot s + c \cdot t + d &= 0, & \tag{11a} \\
e \cdot st + f \cdot s + g \cdot t + h &= 0, & \tag{11b} \\
i \cdot st + j \cdot s + k \cdot t + l &= 0, & \tag{11c} \\
\end{align*}
\]

where

\[
\begin{align*}
a &= (p_{4y} - p_{3y})(v_{2x} - v_{1x}) - (p_{4x} - p_{3x})(v_{2y} - v_{1y}), \\
b &= (p_{4y} - p_{3y})(p_{2x} - p_{1x}) - (p_{4x} - p_{3x})(p_{2y} - p_{1y}), \\
c &= (p_{4y} - p_{3y})v_{1x} - (p_{4x} - p_{3x})v_{1y}, \\
d &= (p_{4y} - p_{3y})(p_{1x} - p_{3x}) - (p_{4x} - p_{3x})(p_{1y} - p_{3y}), \\
e &= (p_{4x} - p_{3x})(v_{2x} - v_{1x}) - (p_{4y} - p_{3y})(v_{2y} - v_{1y}), \\
f &= (p_{4x} - p_{3x})(p_{2y} - p_{1y}) - (p_{4y} - p_{3y})(p_{2x} - p_{1x}), \\
g &= (p_{4x} - p_{3x})v_{1y} - (p_{4y} - p_{3y})v_{1x}, \\
h &= (p_{4x} - p_{3x})(p_{1y} - p_{3y}) - (p_{4y} - p_{3y})(p_{1z} - p_{3z}), \\
i &= (p_{4x} - p_{3x})(v_{2y} - v_{1y}) - (p_{4y} - p_{3y})(v_{2z} - v_{1z}), \\
j &= (p_{4x} - p_{3x})(p_{2z} - p_{1z}) - (p_{4y} - p_{3y})(p_{2x} - p_{1x}), \\
k &= (p_{4x} - p_{3x})v_{1z} - (p_{4y} - p_{3y})v_{1x}, \\
l &= (p_{4x} - p_{3x})(p_{1z} - p_{3z}) - (p_{4y} - p_{3y})(p_{1x} - p_{3x}). \\
\end{align*}
\]

After solving the above equations, the value of $s$ and the collision time $t = t_1$ are obtained. In a similar manner, let Eq.(6b) equal a constant $q$; then, we substitute the derived parametric equations of $x$, $y$, and $z$ into Eq.(6a) with $t = 0$ to acquire the value of $q$ and the collision time $t = t_2$. 
An infinite number of solutions might exist. This means that the two equations of the moving lines are on the same plane, and this situation can be implicitly solved using the dynamic vertex-to-polygon collision test described in the next subsection. Here, we will only consider the situation in which a unique solution is found. By means of Eq.(5) with $t'_a = t_1$, $t'_b = t_2$, $t_a = 0$, and $t_b = 0$, the collision time $t_c$ of the two moving edges can be obtained using

$$t_c = \frac{t_1 \cdot t_2}{t_1 + t_2}.$$  \hspace{1cm} (12)

If $0 \leq t_c \leq 1$, then a collision event has been discovered. The intersection point is computed by substituting $t_c$ for $t$ of Eqs.(6a) and (6b) after some algebraic manipulations. If the point is not located on the two edges (i.e., either of the values of Eqs.(6a) and (6b) is greater than one or less than zero), then the collision event is discarded.

4.3 Dynamic Vertex-to-Polygon Collision Test

Assume that a vertex $p$ initially located at position $p_0$ moves at a velocity $v$ toward a moving polygon consisting of vertices $p_1$, $p_2$, and $p_3$. The velocities of these vertices are $v_1$, $v_2$, and $v_3$, respectively (see Fig. 15). The current position of $p$ is $p_0 + vt$, and the two vectors of $p_2p_1$ and $p_3p_1$ in the polygon are $V_1$ and $V_2$ expressed as:

$$V_1 = p_2 - p_1 + (v_2 - v_1)t$$ \hspace{1cm} (13a)

and $$V_2 = p_3 - p_1 + (v_3 - v_1)t,$$ \hspace{1cm} (13b)

where $t$ is the elapsed time.

![Fig. 15. A moving vertex located close to a moving polygon mutually.](image)

Hence, the normal vector $\mathbf{N}$ of the polygon is computed as follows:

$$\mathbf{N} = V_1 \times V_2$$

$$= \begin{vmatrix}
\mathbf{i} & \mathbf{j} & \mathbf{k} \\
a + bt & c + dt & e + ft \\
g + ht & i + jt & k + lt
\end{vmatrix}$$

$$= m \mathbf{i} + n \mathbf{j} + o \mathbf{k},$$  \hspace{1cm} (14)
where
\[ a = p_{2x} - p_{1x}, \]
\[ b = v_{2x} - v_{1x}, \]
\[ c = p_{2y} - p_{1y}, \]
\[ d = v_{2y} - v_{1y}, \]
\[ e = p_{2z} - p_{1z}, \]
\[ f = v_{2z} - v_{1z}, \]
\[ g = p_{3x} - p_{1x}, \]
\[ h = v_{3x} - v_{1x}, \]
\[ i = p_{3y} - p_{1y}, \]
\[ j = v_{3y} - v_{1y}, \]
\[ k = p_{3z} - p_{1z}, \]
\[ l = v_{3z} - v_{1z}. \]
\[ m = (dl - fj)^2 + (el + dk - ej - fi)t + (ck - ei), \]
\[ n = (fh - bl)^2 + (eh + fg - al - bk)t + (eg - ak), \]
and \[ o = (bj - dh)^2 + (aj + bi - ch - dg)t + (ai - cg). \]

In the following, we will describe the computation steps to obtain the collision time and collision position. In the first step, let \( p \) be located at the initial position \( p_0 \), i.e., set \( t = 0 \), and assume that the current distance from \( p \) to the polygon equals zero, so that:
\[
[p - (p_1 + v_1t)] \cdot N = 0. \tag{15a}
\]
\[
\Rightarrow \begin{bmatrix} p_x - (p_{1x} + v_{1x}t) \\ p_y - (p_{1y} + v_{1y}t) \\ p_z - (p_{1z} + v_{1z}t) \end{bmatrix} \cdot N = 0. \tag{15b}
\]
Then, the resulting equation is obviously in the cubic form: \( a_3t^3 + a_2t^2 + a_1t + a_0 = 0 \). The Bisection method is next adopted to solve the polynomial of order 3 because it is robust to find the roots of equations [18]. Accordingly, we obtain the collision time \( t = t_1 \) when the vertex is resting and the polygon is moving.

The second step is to let the polygon rest, i.e., set \( t = 0 \); then, the normal vector \( N \) of the polygon is \( [m \ n \ o] = [ck - ei \ eg - ak \ ai - cg] \). When the moving vertex runs into the polygon, the time \( t = t_2 \) is derived from the following manipulations:
\[
(p + vt - p_i) \cdot N = 0. \tag{16a}
\]
\[
\Rightarrow \begin{bmatrix} p_x + v_xt - p_{ix} \\ p_y + v_yt - p_{iy} \\ p_z + v_zt - p_{iz} \end{bmatrix} \cdot \begin{bmatrix} m \\ n \\ o \end{bmatrix} = 0.
\]
\[
\Rightarrow t = \frac{m(p_{ix} - p_x) + n(p_{iy} - p_y) + o(p_{iz} - p_z)}{mv_y + nv_y + ov_z}. \tag{16b}
\]

By using Eq.(12), the collision time \( t_c \) can be calculated, and the collision position is determined via further geometrical computations. If \( t_c \) is larger than one or less than zero, then the collision event is discarded. Otherwise, we use the 3-D Cyrus-Beck algorithm mentioned in subsection 3.3.3 to check whether the vertex \( p \) is inside the polygon at time \( t_c \). If \( p \) is outside the polygon at time \( t_c \), then no collision has been detected. Otherwise, a collision event has been found out.
4.4 The Working Procedure

In this subsection, the entire time-continuous collision detection procedure will be described. Suppose that the present time step is small enough to detect the overlap of object pairs. Once the overlap region of a pair of objects is discovered, the 4-D bounding box of each polygon of the two objects is generated. The 4-D bounding box is constructed by comparing the coordinates of the vertices of the polygon in the current and the previous time steps, as Fig. 16 illustrates. All the 4-D bounding boxes of the polygons are then tested against the overlap region. Therefore, those polygons which sweep the overlap region at a given time interval are detected. Based on the aforementioned strategy, our method for determining both the collision time and position of two polygons of different objects can be summarized as follows. First, the overlap test on a pair of polygons is performed. If the overlap region is detected, then the vertex-to-polygon collision test is applied, and the edge-to-edge collision test is subsequently used. Many feasible solutions of the collision time might be found. Of these solutions, the earliest occurring collision time associated with the collision position is taken as the predicted collision event, and the others are discarded.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of the proposed methods, many experiments were performed on the time-interrupted and the time-continuous collision detection. All the testing programs have been implemented in C++ language by using the Microsoft Visual C/C++ 4.0 compiler under the Microsoft Windows NT 4.0 operating system on a personal computer with a Pentium Pro-180 CPU and 64M RAM. Because Windows NT is a multitasking operating system, we used the Win32 API function, GetThreadTimes, to get the CPU time during the execution of the programs [19]. The performance of the algorithms was tested without considering the time of screen I/O, so that all the programs were developed in the console mode of Windows NT [20] to simplify coding.
5.1 Performance Test on Object-to-Object Collision Detection

In the following experiments, comparison of object-to-object collision detection was accomplished using two simulation programs. The first program exhaustively detects all the bounding boxes of objects pair by pair, thus simulating the spatial occupancy enumeration or bounding volume approach, and the second program uses the space cell method, but not for each pair of bounding boxes.

First, the programs randomly generated both the center and radius of a sphere, which constituted an object. The movable objects moved along predetermined paths lasting 1,000 time steps. In these experiments, all the generated objects were set to be movable. Tables 1 and 2 show the experimental results obtained using the exhaustive method and the space cell method, respectively, under the same simulation conditions. From the results of these two tables, we observe that the number of pairs of objects which must be compared using the latter method is much less than that compared using the former method. Consequently, the execution time needed to test bounding boxes using the space cell method is considerably less than that needed using the exhaustive method. As can be seen in Figs. 17 and 18, the differences of the number in compared pairs and the amount of execution time used between the two methods increased exponentially with the number of objects existing in the simulation world.

5.2 Performance Test on Bounding Box and Azimuth-Elevation Map Methods

In this subsection, we will compare the efficiency of filtering out polygons within or crossing an overlap region of two possibly colliding objects represented by bounding boxes with the efficiency of filtering out those of the objects represented by bounding spheres. Fig. 19 shows three experimental objects, one of which is a ball with 1,986 vertices and 2,049 polygons drawn by the authors while the other two were retrieved from the web site: http://www.3dcafe.com. One is an X-wing, consisting of 1,293 vertices and 2,496 polygons.

<table>
<thead>
<tr>
<th>Object numbers</th>
<th>Compared pairs</th>
<th>Detected pairs</th>
<th>Execution time (100ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>45,000</td>
<td>152</td>
<td>4,806,912</td>
</tr>
<tr>
<td>20</td>
<td>190,000</td>
<td>1,095</td>
<td>21,330,672</td>
</tr>
<tr>
<td>30</td>
<td>435,000</td>
<td>2,067</td>
<td>51,173,584</td>
</tr>
<tr>
<td>40</td>
<td>780,000</td>
<td>3,802</td>
<td>98,641,840</td>
</tr>
<tr>
<td>50</td>
<td>1,225,000</td>
<td>5,941</td>
<td>162,834,144</td>
</tr>
<tr>
<td>60</td>
<td>1,770,000</td>
<td>8,532</td>
<td>250,960,864</td>
</tr>
<tr>
<td>70</td>
<td>2,415,000</td>
<td>10,651</td>
<td>360,117,824</td>
</tr>
<tr>
<td>80</td>
<td>3,160,000</td>
<td>13,352</td>
<td>501,921,728</td>
</tr>
<tr>
<td>90</td>
<td>4,005,000</td>
<td>17,062</td>
<td>657,745,792</td>
</tr>
<tr>
<td>100</td>
<td>4,950,000</td>
<td>22,209</td>
<td>868,849,344</td>
</tr>
</tbody>
</table>
Table 2. Object-to-object collision detection using the space cell method.

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

Fig. 17. The number of objects versus the pairs of objects needed to test their bounding boxes using the exhaustive and space cell methods.

Fig. 18. The number of objects versus the execution time needed to test their bounding boxes using the exhaustive and space cell methods.
and the other is a roach composed of 3,285 vertices and 5,491 polygons. In the following experiments, two same objects performed rotating transformations about their centers and moved close to each other. The bounding box and A-E map methods were evaluated by considering the amount of execution time.

Figs. 20(a)–(c) demonstrate the comparison of the execution time required to find polygons within or crossing the overlap region using the two methods for the three illustrated objects. Fig. 20(a) shows the result obtained using the graphical ball as a test object, where two graphical balls have an overlap region in the 70th time step. From this figure, we can find that the efficiency of the A-E map method is better than that of the bounding box method. When we used the X-wing as another test object, as seen in Fig. 20(b), the overlap region was detected earlier using the A-E map method than it was using the bounding box method; however, the efficiency of the former was less than that of the latter before the overlap region of the bounding boxes existed. Fig. 20(c) exhibits the test results for two translating and rotating graphical roaches. Since the roach was a very complicated and asymmetric object, the performance of the A-E map method was always worse than that of the bounding box method. In these experiments, the polygons selected using either the bounding box or the A-E map method could have further studied to determine which pair exactly interpenetrated itself.
Fig. 20. Time required to find the polygons within or crossing the overlap region between (a) two balls; (b) two X-wings; (c) two roaches using the A-E map and bounding box methods.
5.3 Performance Test on Selecting Polygons for Collision Detection

In the following experiments, a divide-and-conquer method for selecting the fewest polygon pairs to facilitate collision detection was tested and compared with a non-divide-and-conquer method. All the experimental objects illustrated in the previous subsection were used. During the tests, two identical objects rotated about their centers and moved toward each other for polygon-to-polygon collision detection. With the divide-and-conquer or non-divide-and-conquer method, the bounding boxes of the two initial lists of polygons within or crossing the overlap region of the two test objects were regenerated in each time step. Table 3 shows the number of the polygons in two roaches, denoted Object 1 and Object 2, found using 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 method. All the selected polygons were finally passed to the polygon-to-polygon intersection test, and the numbers of recursive calls were recorded. We can

Table 3. Comparison of the numbers of polygons found using the non-divide-and-conquer and divide-and-conquer methods.

<table>
<thead>
<tr>
<th>Time step no.</th>
<th>Number of selected polygons</th>
<th>Number of recursive calls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>non-divide-and-conquer method</td>
<td>divide-and-conquer method</td>
</tr>
<tr>
<td></td>
<td>object 1</td>
<td>object 2</td>
</tr>
<tr>
<td>65</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>66</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>67</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>69</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>461</td>
<td>0</td>
</tr>
<tr>
<td>71</td>
<td>653</td>
<td>53</td>
</tr>
<tr>
<td>72</td>
<td>785</td>
<td>570</td>
</tr>
<tr>
<td>73</td>
<td>1,112</td>
<td>1,235</td>
</tr>
<tr>
<td>74</td>
<td>1,346</td>
<td>1,725</td>
</tr>
<tr>
<td>75</td>
<td>1,410</td>
<td>2,037</td>
</tr>
<tr>
<td>76</td>
<td>1,618</td>
<td>2,196</td>
</tr>
<tr>
<td>77</td>
<td>1,641</td>
<td>2,674</td>
</tr>
<tr>
<td>78</td>
<td>1,348</td>
<td>2,759</td>
</tr>
<tr>
<td>79</td>
<td>1,241</td>
<td>2,600</td>
</tr>
<tr>
<td>80</td>
<td>1,044</td>
<td>2,358</td>
</tr>
<tr>
<td>81</td>
<td>764</td>
<td>1,946</td>
</tr>
<tr>
<td>82</td>
<td>651</td>
<td>1,421</td>
</tr>
</tbody>
</table>
Table 4. Comparison of the amounts of execution time used by the non-divide-and-conquer and divide-and-conquer methods.

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

Fig. 21. Time required to detect interference among polygons within or crossing the overlap region using the non-divide-and-conquer and divide-and-conquer methods.
find that the number of pairs of polygons which had to be checked decreased or became zero after the recursive function was performed. Table 4 shows the execution time for all detected pairs in each time step when the bounding boxes of the two objects had an overlap region. An impressive amount of execution time, as demonstrated in Fig. 21, is saved using the divide-and-conquer method for the duration of collision.

5.4 Performance Test on Time-Continuous Collision Detection

In this experiment, the performance of time-continuous collision detection was tested. We used the same simulation conditions as described in subsection 5.3 to verify the effectiveness of the developed algorithms. As shown in Fig. 22, the time-continuous procedure found a collision event after 80.55 time steps, that is, between the 80th and 81st time steps, whereas the time-interrupted procedure lost that event and found another collision event in the 82nd time step for the translated and rotated roaches. It is obvious that the time-continuous procedure needs more execution time to determine the collision time and position. In addition, this implementation also needs extra memory to store the previous positions of the vertices of objects, so that the velocities of the vertices will be made available without extra computing cost.

![Fig. 22. Time required to detect a collision event between two moving roaches using the time-continuous and time-interrupted procedures.](image)

6. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we have developed efficient time-interrupted and time-continuous collision detection procedures. The exact collision position can be determined using the time-interrupted procedure in a prescribed time step and the approximate collision position and time can be determined using the time-continuous procedure.

In the object-to-object overlap test, the space cell method can reduce the number of tests required. In doing so, only those moving objects in the same cell need to be checked. As the experiments described in subsection 5.1 reveal, the computational complexity grows
near linearly as the number of objects increases. The experiments were carried out based on
the assumption of the worst case, in which all objects are moving. When not all objects are
moving, the performance of the space cell method is better.

The A-E map method is designed to speed up the process of finding the polygons
within or crossing the overlap region. It can reduce the execution time used in updating the
bounding volume by employing the bounding sphere approach. When the objects are near
convex in shape, the A-E map method is more efficient than the exhaustive method, which
does not test all the polygons against the overlap region. However, when the bounding
sphere of an object can not compactly encircle its occupied space (i.e., most of the space is
not used by the object), the worst efficiency will result. In future research, we will focus on
solving this problem using a hierarchical scheme and will try to improve the data structure
of the A-E map for rapid searching of the vertices within the overlap region.

The divide-and-conquer method is designed to moderate the number of polygon pairs
which need to be further checked. It can localize the smallest overlap region and leads to
checking the fewest polygon pairs. As the result of the experiments shows, many unneces-
sary tests are eliminated, so the collision detection procedure is accelerated. The algorithm
for our proposed method is simple and easy to implement, and requires less memory stor-
age than does the spatial occupancy enumeration approach based on octrees or BSP trees.

As for time-continuous collision detection, we have proposed a method to simplify
the testing of swept volumes. It has the ability to detect collision events that are missed by
the time-interrupted procedure, as shown in the experiment. For rotating objects,
evertheless, detection errors may occur because we assume that the paths of movement of
the objects are straight, and that the velocities of the constituting vertices are constant.
Further study on the real motions of moving objects is needed. Since all experimental
objects are artificially constructed by model builders, some noises and imperfection in-
volved in real cases can not be considered. To improve the degree of fidelity with the real
world, the analysis of errors due to different modeling details should also be included in
future work.

REFERENCES

2. S. Bonner and R. B. Kelley, “A representation scheme for rapid 3-D collision detection,” in
Proceedings of IEEE International Symposium on Intelligent Control, Arlington,
Virginia, 1988, pp. 320-325.
objects for operator assistance in virtual cooperative workspace,” in Proceedings of the
2nd IEEE International Workshop on Robot and Human Communication, Tokyo, Japan,
1993, pp. 442-447.
4. A. P. del Pobil, M. A. Serna and J. Llovet, “A new representation for collision avoid-
ance and detection,” in Proceedings of IEEE International Conference on Robotics
5. J. H. Youn and K. Wohn, “Realtime collision detection for virtual reality applications,”
in Proceedings of IEEE Virtual Reality Annual International Symposium, Seattle,


Chin-Shyurng Fahn (范欽雄) was born in Tainan, Taiwan, Republic of China, on October 15, 1958. He received the B.S. degree in electronic engineering from National Taiwan Ocean University, Keelung, Taiwan, in 1981, and the M.S. and Ph.D. degrees both in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1983 and 1989, respectively. From August 1983 to July 1984, he served as a Teaching Assistant in the Department of Electrical Engineering at National Cheng Kung University. From August 1983 to July 1989, he worked as an Adjunct Research Assistant in the Institute of Electrical Engineering of that university. From November 1989 to May 1991, he served in the Chinese Army as a Signal Officer. In the meantime, he acted as a Lecturer in the Department of Mechanical Engineering at the Army Engineer School, Kaohsiung, Taiwan.
Since August 1991, he has been an Associate Professor in the Department of Electrical Engineering at National Taiwan University of Science and Technology, Taipei, Taiwan. His current research interests include fuzzy systems, neural networks, and evolutionary computing applied in the areas of image processing, pattern recognition, computer vision, computer graphics, and virtual reality.

**Jui-Lung Wang** (王瑞龍) was born in Chiayi, Taiwan, Republic of China, on September 1, 1971. He received the B.S. degree in electrical engineering from Tamkang University, Tamsui, Taiwan, in 1995, and the M.S. degree in electrical engineering from National Taiwan University of Science and Technology, Taipei, Taiwan, in 1997. From January 1998 to August 1999, he served in the Chinese Army as an Infantry Officer. Now he is a Software Engineer in the Very Innovative Architecture (VIA) Technologies, Inc., Hsintien, Taiwan. His current research interests are in the areas of computer graphics and virtual reality.