Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 8302974, 14 pages
https://doi.org/10.1155/2022/8302974

Research Article

Visibility Detection of 3D Objects and Visual K-Nearest Neighbor Query Based on Convex Hull Model

Tianbao Hao,[1] **Yongshan Liu**,[1] **Xiang Gong**,[2] **Dehan Kong**,[2] **and Jianjun Wang**[1]

[1]*Department of Information Science and Engineering, Yanshan University, Hebei Street No. 438, Qinhuangdao 066004, Hebei, China*
[2]*Department of Information Engineering, Hebei University of Environmental Engineering, Jingang Street No. 8, Qinhuangdao 066102, Hebei, China*

Correspondence should be addressed to Tianbao Hao; 283454988@qq.com

Received 25 October 2021; Revised 2 April 2022; Accepted 27 May 2022; Published 16 June 2022

Academic Editor: José García

With the rapid development of the technologies of virtual reality (VR) and augmented reality (AR), the accurate visual query of 3D objects which are closer to real world has become a hotspot of study. There are a lot of redundancy problems in the existing axis-aligned bounding box model (AABB) and oriented bounding box model (OBB), which are used to represent 3D objects mostly at present, and cannot be used for the accurate visual query. For example, the results of the visual query must be accurate, and without any error in automatic driving technology in the automotive industry, otherwise, serious safety problems would be caused. The convex hull model could be used to solve those problems. However, there are few studies on visual query based on the convex hull model. A visual query method for 3D obstacle space based on the convex hull model is proposed in this paper. Firstly, the definitions of "query point-object visual body" and "combined obstacle" are proposed. Secondly, the visibility detection algorithm of the convex hull model is given in detail. The collision detection method of "query point-object visible body" combined with other obstacles and line-plane geometric calculation operations are used to detect the visibility of the convex hull model. Then, a 3D visual k-nearest neighbor algorithm based on the convex hull model is given in the paper. Finally, the algorithm is verified by experiments and compared with the traditional visibility detection algorithm, and the analysis of experimental results shows that the algorithm has a great performance and higher accuracy. The growth rate of the query time is smaller, and the speed is faster; especially, when there are fewer query points, the query speed can be increased by more than 50%.

1. Introduction

In recent years, with the rapid development of computer graphics, human-computer interaction technology, sensor technology, and network technology, and VR and AR technology have been developed rapidly and used widely in various fields. For example, in the medical field, virtual human organ models [1] with the same proportions of real people can improve the pertinence and effectiveness of doctors' training, which could improve the accuracy of surgical, and eliminate the risk of being blamed morally for repetitive practice on real people. In addition, it is used in mechanical surgery to achieve accurate tumor resection. In the military field, VR can also be used to train military personnel to explore and learn new skills. The US. Department of Defense has developed multiple virtual battlefield systems [2], which can realize exercises that multiple people work together in virtual space, precision strikes, and radar detection of fighter jets can be achieved. In the aerospace field, NASA has developed a VR system to improve the proficiency of astronauts participating in missions in outer space [3]. In the automotive industry, the application of autonomous driving technology can enable cars to avoid obstacles during autonomous driving and find the optimal driving path. All of the above belong to the research of visual query in three-dimensional obstacle space. In VR applications, invisible spatial objects are invalid data for the users. Based on this principle, the visual query in

three-dimensional obstacle space is a research content with high academic and commercial value. However, there are still scarce studies in this field.

Visual query is a kind of query in obstacle space, and it faces unique problems in three-dimensional space, for example, the problem of building a 3D space object model. At present, some achievements have been made in the research of spatial database in two-dimensional space, such as storage and index structure of objects and obstacles, prediction and storage of viewable fields, visibility detection of object, and calculation of visual subinterval in continuous visual query. However, the research on visual query of three-dimensional objects is just emerging. Different from the general query in obstacle space, it is closer to the real world. While the objects in the three-dimensional space have the characteristics of uneven distribution density, irregular shape, and a large amount of the data, for the visibility detection of the real 3D objects, the traditional double projection method cannot be used as the detection of AABB bounding box, nor the detection method of "based on horizontal projection angle curve" which is projected to the horizontal plane and vertical plane like the detection of OBB bounding box [4]. The object model is arbitrary and close to reality in three-dimensional space, which will lead to the lack of a unified projection plane among the query objects, data objects, and obstacle objects, and the visual relationship cannot be represented correctly in the projection by the positions of the three types of objects. In order to solve these problems, a visual query method based on the convex hull model in three-dimensional obstacle space is proposed, which fills the gap of current research by enabling accurate query of objects when multiple obstacles combine together to block the visible area. It has important value in the automation industry.

Compared with the existing studies, the contributions of our research are highlighted as follows: firstly, the GJK collision algorithm, which is introduced in Section 3.1, is used to detect the visibility of the three-dimensional space, which can filter out a lot of objects and improve the efficiency of the algorithm. Secondly, according to the complex situation of obstacles in three-dimensional space, the obstacles that collide with each other are combined to form combined obstacles, and the visual detection algorithm is performed. Finally, the real three-dimensional space is simulated, and the k-nearest neighbor query algorithm is proposed. Due to the convex hull model, the query accuracy is higher, and the query speed is faster than the traditional methods in 3D obstacle space.

The structure of the paper is as follows: in Section 1, an overall introduction and the significance of the research are described. In Section 2, a brief summary of related work is given. In Section 3, some definitions, such as "query point-object visual body" and "combined obstacles", and related theorems are given. In Section 4, a detailed introduction and description of the visibility detection algorithm are given. In Section 5, the visual k-nearest neighbor query algorithm based on a convex hull is introduced in detail. In Section 6, the experimental environment is set up, and an experimental comparison is made to prove that the query algorithm has a good performance.

2. Related Works

There had been a lot of research on the visibility detection of three-dimensional objects, and some progress had been made. In 2017, Alipour et al. [5] studied the visibility counting problem (VCP) and visibility testing problem (VTP) of data in a 2.5D terrain environment and gave an approximate algorithm for visibility calculation and detection, which reduced the computing cost. Chen and Wang [6] studied the calculation of weak visibility between line segments in simple polygons. Zarei and Ghodsi [7] proposed a method to solve the problem of query point visibility computation in polygons with holes. Ghodsi et al. [8] gave a method of α-visibility based on the concept of α that could be applied to weak visual detection between point and line segment and between line segment and line segment. Arkin [9] obtained the shortest path *fro*m point S to the query point q within the time of $O(Klog^2n)$ by using a data structure in which time complexity was $O(n^2 2a^{(n)} logn)$, $a(n)$ was the inverse Ackermann function, and K was the number of query points. Wang H [10] proposed a new algorithm for the shortest path segmentation query problem to optimize the query result of the visibility of the polygon area and use the data structure in which time complexity of construction was $O(nlogh + h^2)$ and query time was $O(hloghlogn)$. When h was small relatively, the overall performance improved greatly. The visual query researches mentioned above-required precalculation of the visual area, which could not meet many requirements of the practical applications.

Therefore, extensive and in-depth researches that did not require precalculation had been conducted on visibility queries. Shou et al. [11] proposed an HDoV tree to improve the performance and visual fidelity of the visualization system. Kofler et al. [12] combined R-tree and LOD and proposed a LOD-R tree index structure that could handle large amounts of data effectively. Zhang et al. [13] discussed the problems of spatial query processing when obstacles existed and proposed an integrated framework that could solve these problems effectively. Gao et al. [14] proposed a new query algorithm for the reverse visual k-nearest neighbor query to deal with the visibility problems of the reverse k-nearest neighbor query in two-dimensional space. Yu et al. proposed a reverse k-nearest neighbor query method in obstacle space [15] and proposed a continuous reverse k-nearest neighbor query subsequently based on controlled points [16]. These methods were still not suitable for visual query in three-dimensional space. Zhang et al. [17] combined the characteristics of the Voronoi diagram and the corresponding pruning algorithm to optimize the group reverse k nearest neighbor query and proposed the V_GRkNN algorithm. Song et al. [18] proposed the SOS algorithm and the ADD_SOS algorithm to solve the skyline query problem in the obstacle space. In addition, the research on the visual query of moving objects was also an important branch of visual queries. Yu et al. [19] proposed a DSI index structure that could process k-NN queries in a distributed manner. Han et al. [20] proposed a new GRI framework and H-GTPR tree index structure for the

problem that moving objects on the urban road often changed, and it was not easy to track and query. The studies above were more suitable for centralized data processing. Cho et al. [21] proposed a distributed expansion scheme of DAEMON, which could be used for CkNN query in the road network. Vorona et al. [22] proposed an approximate geospatial query processing engine based on deep learning, which could respond to flexible aggregation queries, but the study did not consider the issue of visibility. Wan et al. [23] proposed a COPkNN query to solve the continuous k-nearest neighbor query problem of moving objects. Li et al. [24] studied the query algorithm of indoor obstacle space for the problem of moving objects affecting each other's deformation properties and proposed the DSP-Topk query algorithm. Compared with the existing Topk algorithm, it was only suitable for a small number of target objects. Yu et al. [25] proposed a query method based on a moving window to search random trees quickly and dealt with the problems of path planning in obstacle spaces. Luo et al. [26] studied the visual nearest neighbor query of moving objects on continuous trajectories.

The studies mentioned above were in the two-dimensional space, and their results could not be applied directly to the three-dimensional visual query. The visibility detection method of objects and the calculation of obstacle distance in three-dimensional space were more complicated than those in two-dimensional space. Arman et al. [27] proposed an efficient and scalable interactive system VizQ to deal with the visibility query problem in the three-dimensional obstacle space. However, the system was based on the AABB model, which had a low reliability of the visibility detection algorithm. In addition, some scholars studied moving objects in 3D visual queries. Haider et al. [28] studied the methods of finding out the best viewpoint continuously from a set of candidate viewpoints and proposed a k-continuous maximum visibility query algorithm (kCMN), which could provide the best line of sight for moving objects in 2D or 3D obstacle space. Rabban et al. [29] designed an effective graph theory method for the maximum visibility facility selection query (MVFS), which solved the discrete space problem, but did not support the applications of the new database. The idea of vertex separators was used to design an effective graph theory method. A large number of obstacles and facility locations were expanded. With the continuous development of geographic information systems, a 3D spatial data model was used to solve data problems collected by visualization tools, such as mechanical properties of rock mass [30], heterogeneous slopes [31], elastic properties of rock materials, and plastic yield stress problem [32], and the distribution of particulate material between gaps [33]. New applied models had also been created to address these practical problems, 3D multiphase meso-models for the case where concrete damage and transport properties were coupled [34]. Contact detection algorithms based on Minkowski differences are applied to convex polygon/polyhedron discretization/in discontinuous modeling [35]. This method did not consider the presence of concave bodies. Meng Qingxiang proposed a three-dimensional mesoscale computational model of concave granular soil-rock mixtures [36] and conducted research on concave bodies in 3D space. The calculation method was complex and suitable for processing a small number of objects.

In summary, many data models and research methods had been proposed according to practical problems. However, the methods mentioned above were not suitable for accurate query in three-dimensional space. At present, the research on the query in three-dimensional space is just beginning. Combining the characteristics of the convex hull model and the characteristics of vision space, the visibility detection algorithm and visible k-nearest neighbor query algorithm are proposed based on the convex hull model.

3. Related Definitions and Theorems

Compared with the traditional three-dimensional obstacle space, the main difficulty in the three-dimensional obstacle space of this paper is that both the data object and the obstacle object are three-dimensional objects represented by the convex hull model, and there is no unified projection plane for them. Therefore, the visibility cannot be detected by the projection method, nor can the detection method "based on the horizontal projection angle curve" be used. In addition, the obstructed areas of the obstacle objects in the three-dimensional space are three dimensional, so the viewable area cannot be calculated by the angle like those in the two-dimensional space. In this paper, the visibility detection of three-dimensional objects is regarded as the detection of the spatial position relationship between the visible objects and the obstacles. The space geometry algorithm of GJK collision detection is used to calculate the spatial position relationship between the data object and the obstacle which are represented by the convex hull model, and the initial shielding relationship is obtained between the query object and the obstacle. Then, we use the line-surface relationship geometry calculation to further detect the visibility of the three-dimensional object and complete the visibility detection of the three-dimensional object based on the convex hull model.

3.1. Overview of GJK Algorithm. One of the most effective algorithms for the intersecting polyhedrons is an iterative algorithm developed by Gilbert, Johnson, and Keerthi, known as the GJK algorithm for short. The algorithm defines two sets of the vertices and calculates the Euclidean distance between the elements of the sets. It should be emphasized that the GJK algorithm does not perform any substantive operations on the set of input points but calculates the Minkowski differences between the points. The related problem is simplified from calculating the distance between two sets of convex bodies to solving the distance in a single set of convex body. The GJK algorithm searches for the Minkowski difference object and obtains a subspace volume during each iteration of the loop, and the spatial volume is a singular body. In addition, the algorithm does not display the calculation of the Minkowski difference and adopts a sampling calculation for the Minkowski difference according to specific requirements and through a support mapping

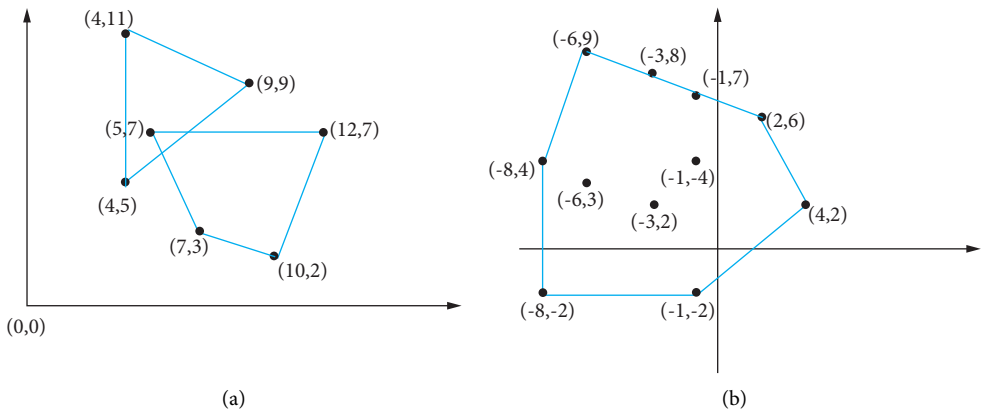

Figure 1: GJK algorithm. (a) Intersection of two convex bodies. (b) Minkowski difference.

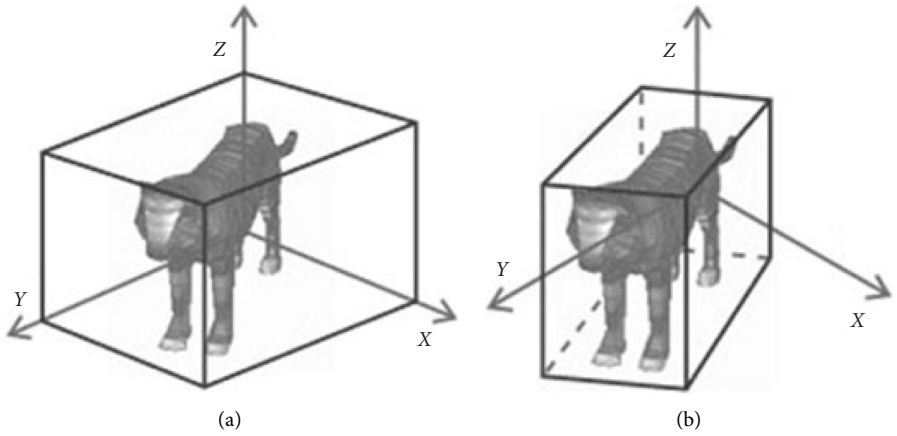

Figure 2: AABB model and OBB model. (a) AABB. (b) OBB.

function. The algorithm is based on the following principle: as shown in Figure 1(a), the distance between the two polyhedrons A and B is equivalent to the distance between the Minkowski difference $C(C = A\theta B)$ and the origin, as shown in Figure 1(b). Therefore, the problem is simplified to solve the nearest point on C from the origin. The explicit calculation of the Minkowski difference is quite complicated. GJK algorithm adopts the method of sampling the vertex set of the Minkowski difference through the support mapping function $C = A\theta B$ to simplify the operation. In particular, the support function $S_A(d)$ maps the given direction d to a certain support point in the convex object A in that direction. Since the supporting function reflects the maximum value of the linear function, the supporting mapping for C is $S_{A\theta B}(d)$, where the correlation mapping between A and B can be expressed as $S_{A\theta B}(d) = S_A(d) - S_B(d)$. Therefore, the corresponding vertices of the Minkowski difference can be calculated from polyhedron A and B according to the specific requirements.

3.2. The Related Definitions

Definition 1. AABB model: AABB is short for axis-aligned bounding box model. It is defined as the smallest hexahedron that contains the object and whose sides are parallel to the coordinate axis. The structure of the AABB model is simple relatively, with small storage space, but poor compactness. Especially, for elongated objects placed diagonally, the redundant space is large as shown in Figure 2(a).

Definition 2. OBB model: OBB is short for oriented bounding box model. It is the smallest rectangular parallelepiped that contains the object and is relative to the coordinate axis in any direction as shown in Figure 2(b). The spatial redundancy of the OBB model is smaller than the AABB model.

Definition 3. 3D spatial object: the object in 3D space is a polyhedron represented by a 3D convex hull. Each convex hull polyhedron is represented by a $List(C)$, and $C=(X, Y, Z)$ is a coordinate point constituting the surface of the polyhedron as shown in Figure 3.

Definition 4. Shortest distance: the distance between the query point q and the three-dimensional object represented by convex hull polyhedron should be considered separately: the distance between q and vertex, q and edge, q and surface.

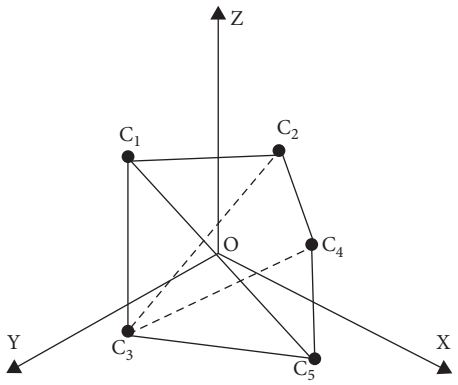

Figure 3: Object in 3D space.

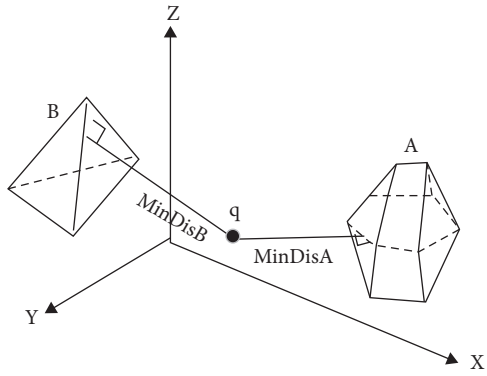

Figure 4: The shortest distance.

After comparison, the shortest distance is obtained from the query point q to the polyhedron, which can be represented as the shortest distance from the query point to the convex hull, as shown in Figure 4.

Definition 5. Visibility of 3D spatial object: given a query point q, set Obj as an object set in the 3D space and Obs as an obstacle set in the 3D space. If $\exists obj \in Obj$ is visible to the query point q, it means that there is at least one point o on the object obj, and the line between o and q does not pass through any obstacle, which means $\forall obs \in Obs$, $o \in obs$, $[o, q] \cap obs = \varnothing$.

Definition 6. Query point-object visual body: given a query point q, a three-dimensional spatial object set Obj and a three-dimensional obstacle set Obs, assuming that $\exists obj \in Obj$ and the query point q form a three-dimensional viewing cone, then the convex hull model regenerated after adding the coordinate point of the point q to the coordinate set of the object obj is called the "query point-object visual body" as shown in Figure 5.

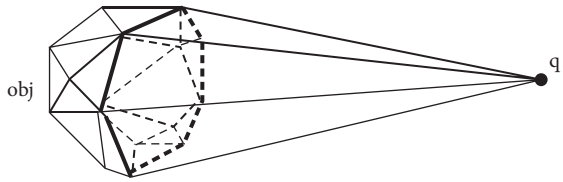

Figure 5: Query point-object visual body.

Definition 7. Combined Obstacle: given a query point q, a three-dimensional object obj and a three-dimensional obstacle set Obs, assuming $\exists obs1 \in Obs$, $\exists obs2 \in Obs$, ..., if $obs1$, $obs2$, ... have a collision relationship, then the smallest outsourcing polyhedron of these obstacles $obs1$, $obs2$, ... is the "combined obstacle". As shown in Figure 6, the two obstacles $obs1$ and $obs2$ collide, and the smallest outsourcing polyhedron $obs12$ formed by $obs1$ and $obs2$ is the "combined obstacle".

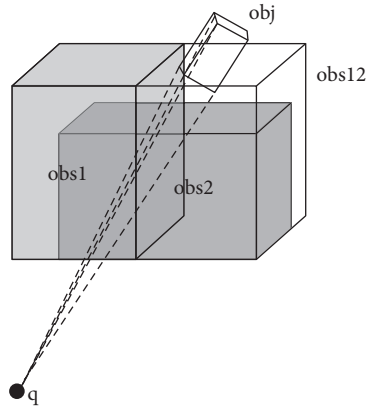

Figure 6: Combined obstacle.

3.3. The Related Theorems

Theorem 1. *Given a three-dimensional spatial object set O, a three-dimensional obstacle object set O' and a query point q. For any $o \in O$, $o' \in O'$, if there is a collision between o' and the "query point-object visual body" q-o, then there is an occlusion relationship between o' and o.*

Prove: As shown in Figure 7, the query point q and the spatial object o in the three-dimensional space formed a "query point-object visible body" q-o, and it collides with an obstacle o'. There must be a ridge line on the q-o passing through the obstacle o', which means that a part of the line of sight from the query point q to the spatial object o is projected by the obstacle o', the line of sight is blocked, and the obstacle and the object have an occlusion relationship.

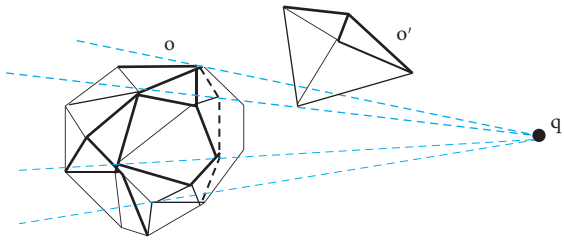

Figure 7: A common partial occlusion relationship.

Theorem 2. *Given a three-dimensional object set O, a three-dimensional obstacle set O' and a query point q, for any*

$o \in O$, $o' \in O'$, q and o form a "query point-object visual body" q-o, q and o' form a "query point-obstacle visual body" $q - o'$. If all the outermost edges of q-o including the query point q can pass through all the outermost faces on $q - o'$, then o' obscures o completely, and the query object o is invisible to the query point q.

Proof. The occlusion relationship between the object and the obstacle can be regarded as the positional relationship between the outermost edges of the "query point-object visible body" q-o and the outermost surface *Facs* of the "query point-obstacle visible body" $q - o'$. As shown in Figure 8, the reason for adopting the outermost edges of q-o including the query point q and the outermost outer surface *Facs* of $q - o'$ for detection is that the outermost edges of q-o can form the largest visual field of the object o. If the largest field of view of the spatial object based on the query point is occluded completely, the smaller field of view formed by the internal point set must also be occluded completely. There is no need to test the smaller field. If the smaller field of view formed inside is obscured completely, it is not certain whether the entire field of view of the spatial object relative to the query point is obscured. Moreover, the number of smaller internal fields of view is large, the formation methods are complicated, and their calculations should waste a lot of time. So, if all the outermost edges containing q on q-o can pass through the face *Fac* formed by q and $q - o'$, it means that the object's maximum field of view relative to the query point is blocked, and the spatial object is invisible to the query point q. □

4. Visibility Detection of Convex Hull Model

According to the definitions and theorems in the last section, the visibility detection algorithm based on the convex hull model is introduced below, which will be used for the visual k-nearest neighbor query. The visual k-nearest neighbor query treats all the objects in front of the current query object as obstacles. It can be seen that the number of obstacles for the current query object is variable, and there may be one or more obstacles. The number of obstacles that can block the current query object is also variable. There may be one or more obstacles, and the query object representation model is a convex hull model.

For the visibility detection of a single obstacle, the positional relationship of the outermost edge of the "query point-object visible body" and the surface of the "query point-obstacle visible body" is used directly to make judgments. The specific process of the visibility detection algorithm for obstacles represented by a single convex hull model is as follows:

The algorithm is described as follows. The query point q is added to the query object point set to form the original point set of the "query point-object visible body" *qObjHu*. The outermost edges *qToridge* containing the query point q of the convex hull model are obtained (lines 2–4). The original point set O of the obstacle is used to construct the convex hull model *ObsHull*, and the surface triangle *fac* is obtained (lines 5-6). If there is a ridge line that does not pass through any surface, then the query object represented by the convex hull model is visible, and the result True is returned. Otherwise, it is invisible, and the result *False* is returned (lines 13–17).

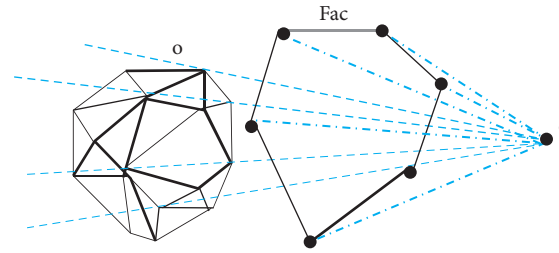

Figure 8: Complete occlusion relationship.

For the visibility detection of multiple obstacles, we have to judge the relationship between the obstacles. The line-surface position detection method is used to make the judgment.

The "query point-obstacle visual bodies" between obstacles adopt the line-surface position detection method instead of the collision detection method between obstacles as is shown in Figure 9. There is no collision between obstacle *Obs1* and obstacle *Obs2*, and both obstacles have an occlusion relationship for the query object, but the query object is invisible to the query point q. This is because the set of obstacles formed by *Obs1* and *Obs2* obscures the query object, the straight line from point q to the query object *obj* is occluded by the "combined obstacle" of *Obs1* and *Obs2*.

The "query point-obstacle visual bodies" between obstacles adopt the line-surface position detection method instead of the collision detection method between "query point-obstacle visual bodies" based on obstacles as shown in Figure 10. The gray area in the figure is the visible area based on the point q. Obstacle *Obs1* and *Obs2* form the "query point-obstacle visual bodies" q-*Obs1* and q-*Obs2*, respectively. If the collision detection method is used to detect whether q-*Obs1* and q-*Obs2* collide, the result obtained by this method is that the two objects always collide, because q-*Obs1* and q-*Obs2* have a common point q, which will inevitably lead to a collision. However, the "combined obstacle" formed by them blocks the visible area (gray area), and the line-surface position detection method will make the result more accurate and will not make wrong judgments because of the existence of q.

The construction method of "combined obstacle" in three-dimensional space will be used to obtain the combined obstacles in the visual k-nearest neighbor query, and the positional relationship between multiple obstacles includes the following two types.

(1) The obstacle *obs* does not collide with other "query point-obstacle visual bodies" based on other obstacles. Then, only the "query point-object visual body" based on the query object needs to be detected by the positional relationship between the outermost edge of the visible body and the outermost surface of obstacle *obs*.

(2) The obstacle *obs* collides with other "query point-obstacle visual bodies" based on other obstacles. We

```
Input: query point q, obstacle point set O, query object set Obj
Output: True/False
(1)  BEGIN
(2)  lis_qObj←Obj.Add(q);
(3)  qObjHu←construct(lis_qObj);
(4)  qToridge←getq_ridge(qObjHull);
(5)  ObsHull←construct(O);
(6)  fac←getFac(ObsHull);
(7)  for each line in q_ridge do
(8)     for each f in fac do
(9)        num←deal (line, f);
(10)       lis_View.Add(num);
(11)       end for
(12) end for
(13)    if (lis_View.Cotains(0)) then
(14)       Return True;
(15) else
(16)       Return False;
(17) end if
(18) END
```

Algorithm 1: OneJudVisible algorithm.

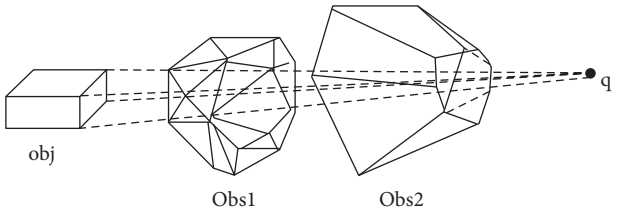

Figure 9: A positional relationship between obstacles.

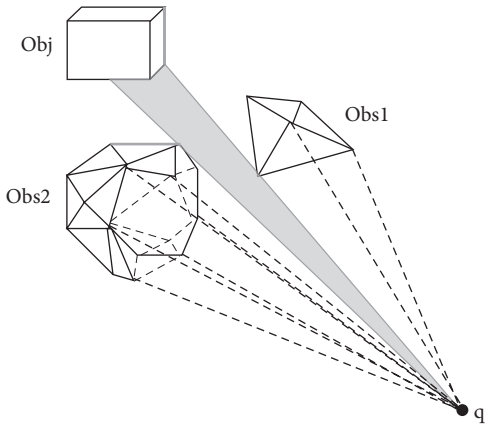

Figure 10: The positional relationship of "query point-obstacle visual bodies" and "query point-object visual body."

combine the point set of obstacle *obs* and the point sets of other obstacles which form the "query point-obstacle visual bodies" and collide with the obstacle *obs* to form a whole convex polyhedral obstacle set. Then, only the positional relationship between the outermost edge line containing q and the outermost outer surface of the convex polyhedron needs to be detected.

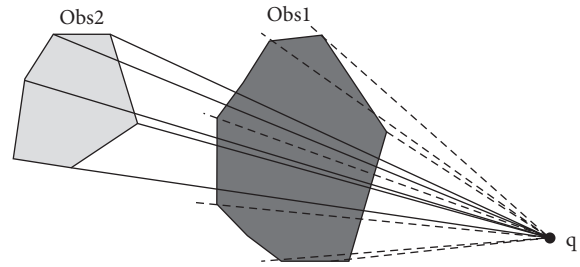

Figure 11: T obstacle occludes another obstacle completely.

The positional relationship between obstacles may be that the obstacle in front occludes the obstacle completely in the back. Then, the results of "query point-obstacle visual body" based on the obstacles and line-surface position method detection are all "no ridge line passing through the obstacle surface," but, at this time, the formation of combined obstacles has no effect on the result as shown in Figure 11.

The cross sections of obstacle *Obs1* and obstacle *Obs2* are relative to the query point q. There is no case where outermost edge lines containing q of *q-Obs1* pass through the obstacle *Obs2*, but this situation does not affect the obstacle collection formed by obstacles and the visibility detection of the query object, because the occlusion of the query object by the obstacle *Obs2* is equivalent to the "combined obstacle" of obstacle *Obs1* and obstacle *Obs2*.

The algorithm is described as follows. Firstly, the point set of all obstacles is processed. Each original point set of the obstacle is added the query point q to form a "query point-obstacle visual body" based on the obstacle set O (line 2). A set *ListExitIndex* is initialized to store the obstacle ids of the established combined obstacles (line 3) and then loops through the set of all obstacles to check whether the current obstacles have established "combined obstacles" (lines 4-5).

```
    Input: query point q, obstacle point set List<Obs> O
    Output: "Combined Obstacle" Set List<Obs> O'
(1)  BEGIN
(2)  ListObs←Construct(q, O);
(3)  ListExitIndex = null;
(4)    for(k = 0; k < ListObs.size(); k++) do
(5)      if(!ListExitIndex.Contains(k)) then
(6)        ListExitIndex.Add(k);
(7)        obs = ListObs[k];
(8)      end if
(9)      for(i = 0; i < ListObs.size(); i++) do
(10)       lines = getLines(obs);
(11)       face = getFace(ListObs[i]);
(12)       if(isThrough(lines, face)) then
(13)         obs = ComObs (obs, listObs[i]);
(14)         ListExitIndex.Add(i);
(15)       end if
(16)       O'.Add(obs);
(17)     end for
(18)   end for
(19)   Return O';
(20)   END
```

Algorithm 2: MulObs algorithm.

We establish "combined obstacle" for current obstacles if there is no one. It is divided into three steps. The first step is to obtain the outermost edges of the current obstacle (line 10). The second step is to obtain the outermost surface *face* of each of the "query points-obstacle visual body" *ListObs* based on the obstacle set (line 11). The third step is to judge whether the lines of the current obstacle pass through the face of each obstacle. If there is a passing relationship, it means that the two obstacles can form a "combined obstacle". We add the obstacle point set which collides with it to the combined obstacle set (lines 12-13) until all obstacles are judged. Then, we add the current obstacle to the set of "combined obstacle" (line 16), and finally, a complete set of combined obstacles for the current obstacles is obtained (line 19).

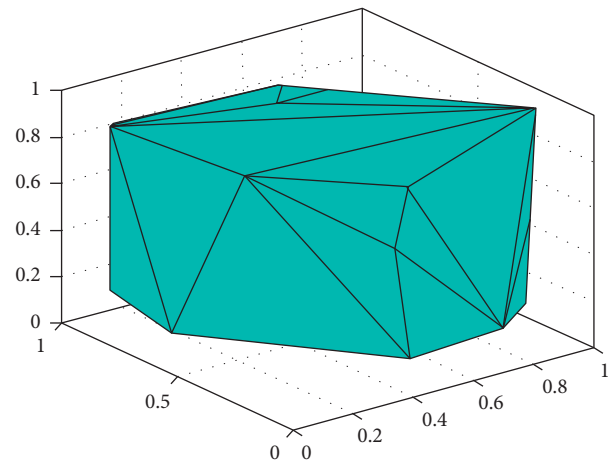

Figure 12: The convex hull in the three-dimensional space composed of triangles.

5. Visual k-Nearest Neighbor Query of Convex Hull Model in 3D Spatial Space

The visual k-nearest neighbor algorithm of the convex hull model is processed as follows, which is divided into three steps. Firstly, the spatial objects are sorted by the distance to the query point from the nearest to the farthest. Secondly, collision detection is performed on the "query point-obstacle visible body" formed by the query object and the obstacle. Finally, the occlusion relationship of the query object and the obstacle is detected, and the complete or partial occlusion relationship is obtained. The spatial objects are judged in turn until the visual k-nearest neighbor query result that satisfies the condition is obtained.

5.1. The Shortest Distance of 3D Objects. The shortest distance from a point to a volume calculated in the three-dimensional space is the minimum of all the distances from the point to the vertex of the body surface, the point to the edge of the body surface, and the point to the face of the body surface. The surface of the three-dimensional convex hull is a closed bounding box composed of triangles, as shown in Figure 12.

Each triangle on the surface of the convex hull is represented by coordinates of three points. Through these coordinates, the plane equation of the triangle can be obtained, and the projection point from the point to the plane of the triangle can be obtained. If the projection point is inside the triangle (including the point inside the triangle, the point on the line segment, and the vertex of the triangle), the shortest distance from the point to the triangle is the length of the line segment formed by the point and the

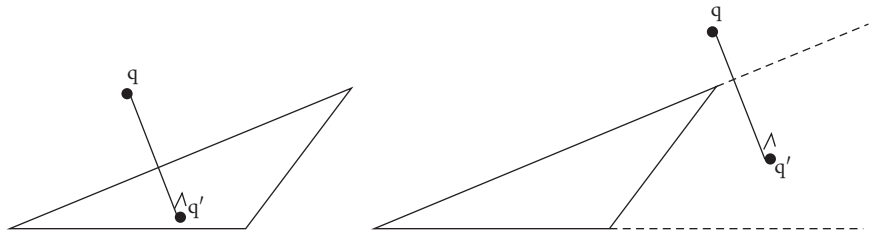

Figure 13: The relationship between the point-to-surface projection and the triangle.

projection point. Otherwise, we calculate the distance from the point to each vertex of the triangle. After comparing the distances of the three vertices, the minimum value is the shortest distance from the point to the triangle, as shown in Figure 13.

The shortest distance calculation algorithm is as follows, which is used to calculate the shortest distance for visual k-nearest neighbor query.

This algorithm is used to calculate the shortest distance between the query point and the convex hull model of the current spatial object. The description of the algorithm is as follows. Firstly, the original convex hull point set O is used to construct the convex hull *Hull*, and the outermost surface *fac* is obtained (lines 2-3). Secondly, the minimum distance between the query point and the surface of the current convex hull is calculated. The specific method is as follows, and we calculate the distance from the query point to each triangular surface of the convex hull surface. If the query point is located in the visible direction of the current triangle surface, we calculate the shortest distance between the triangle and the query point (lines 4–6). Then, we compare the shortest distance between all the surfaces of the convex hull and the query point, and the minimum value is the shortest distance between the current convex hull and the query point (lines 7-8). Finally, the result of the shortest distance is returned (lines 11-12).

5.2. Occlusion Judgment of Obstacle Set. Before the visual k-nearest neighbor query algorithm based on the convex hull model, occlusion judgment of the obstacles is considered firstly in 3D obstacle space. The obstacle set in the visual k-nearest neighbor query is a collection of obstacles, not a single obstacle.

The occlusion judgment algorithm is described as follows. Firstly, the query point q is added to each obstacle point set that collides with the "query point-object body" based on the target object to form the "query point-obstacle body" based on the obstacle (lines 2-3), and the *Concave* and *Ridge* of each "query point-object visual body" are obtained (line 3-4). These "query point-obstacle visual bodies" are based on Theorem 2: if there is an outermost edge line containing the query point of the "query point-obstacle visual body" based on the obstacle, *Obs1* can pass through the outermost surface of the other obstacles, and *Obs1* and the other obstacles can form a convex polyhedron. We make the judgment for all the obstacles, a convex polyhedron is formed (line 4), and check whether there is an outermost edge line containing the query point on the "query point-obstacle visible body" based on the current obstacle passing through the polyhedron outer surface (line 9), if exists, add it to the obstacle set (line 10–14), until the relationships between all other obstacles and the current obstacle have been judged, then the current obstacle point set is added to form a combined obstacle set (line 15). Finally, it is judged whether the set of obstacles occludes the query object completely, and returns the result (lines 18–24).

5.3. Visual k-Nearest Neighbor Query for 3D Objects. After the visibility detection algorithm and the shortest distance calculation algorithm based on the convex hull model, the visual k-nearest neighbor query is to proceed. The following is the specific process of the three-dimensional visual k-nearest neighbor query algorithm based on the convex hull model.

The algorithm is described as follows. Firstly, according to the rule of "distant objects cannot affect the visibility of close objects" in the three-dimensional space, the spatial objects are sorted according to the distance between the spatial objects and the query point in ascending order (line 2). Secondly, the query point is added to the surface point set of the target object, the query point and the target object form a "query point-object visual body" (line 4), and the spatial objects in front of the current query object are all obstacles. The collision detection is performed between the "query point-object visual body" and the objects in the obstacle set (lines 5–7). If the "query point-object visual body based on the query object does not collide with any obstacle, it will be added to the result set and jump out of the current loop to continue the visibility detection of the next query object (lines 12–14). If the collision occurs, the obstacle that collided with the obstacle candidate set *ObsSet* is added to judge whether the obstacle in *ObsSet* occludes the query object completely (line 15), if it does not occlude the current query object completely, jump out of the current loop, and judge the visibility of the next query object, otherwise jump out of the current loop directly and determine the visibility of the next query object (line 15–17). Finally, the visual k-nearest neighbor query results which meet the conditions are obtained (line 21).

6. Experimental Results and Analysis

6.1. Experimental Environment Settings and Data. This experiment was under the Microsoft Windows 10 operating

```
Input: query point q, convex hull point set O
Output: result minddl
(1)  BEGIN
(2)  Hull←construct(O);
(3)  fac←getFac(Hull);
(4)    for each f in fac do
(5)      if(q.view(f) = = true) then
(6)        discurl←dl(line, q, f);
(7)        Discurls.Add(discurl);
(8)      end if
(9)  end for
(10) discur←Discurls.getMin;
(11) minddl←ddl.Add(o.id, discur);
(12) Return minddl;
(13) END
```

Algorithm 3: MinDistance algorithm.

```
Input: query point q, candidate set Obj, obstacle set ObsSet;
Output: True/False;
(1)  BEGIN
(2)  for each obs in ObsSet do
(3)    pToObs←Add(q);
(4)    Concave←BuildConcave (pToObs);
(5)    Ridge←GetRidge(Concave);
(6)    Fac←GetFac(Concave);
(7)    for each ridge in Ridge do
(8)      for each fac in Fac do
(9)        if (ridge.id ! = fac.id) then
(10)         num←IntSegTri (ridge, fac);
(11)       end if
(12)     end for
(13)   end for
(14)   if num.Contains(1) then
(15)     Lis_Obs.Add(ObsSet [ridge.id]);
(16)   end if
(17) end for
(18) mulObs←MulJud (q, Obs, Lis_Obs);
(19) TorF←OneJudVisible(q, Obj, mulObs);
(20) if TorF = = false
(21)   Return True;
(22) else
(23)   Return False;
(24) end if
(25) END
```

Algorithm 4: Occlude algorithm.

system based on a 64-bit processor, the development environment was Visual Studio Enterprise 2015 and MatLab, the main program programming language was C#, and the hardware environment was Intel(R) Core(TM) i5-4200M CPU @2.50 GHz, quad-core, 8 GRAM.

The *SpatialDataGenerator* software is a spatial data generation tool that can be downloaded for free from the Internet. This paper used the software to generate random objects in three-dimensional space. These objects were all axis-aligned bounding boxes with regular shapes. However, the represented model of the spatial objects in the paper was a three-dimensional convex hull. For comparative experiments, we needed to generate the coordinates of the point set of the convex hull in the axis-aligned bounding box corresponding to the same position. The data set generated method adopted in the experiment was as follows. The extreme point coordinates of the axis-aligned bounding box of each three-dimensional spatial object were generated by *SpatialDataGenerator* software. We used the extreme point value of each bounding box as the interval range, and a

```
Input: query point q, date set O, value k;
Output: result set Visual;
(1)  BEGIN
(2)  ddl←Sort(O, q);
(3)  for each Obj in ddl do
(4)    qToObj←Add(q);
(5)    for each Obs in ddl do
(6)      if Obs.address < Obj.address then
(7)        TorF←GJK(qToObj, Obs);
(8)      end if
(9)      if (TorF = = true) then
(10)       ObsSet.Add(Obs);
(11)     end if
(12)     if ObsSet.Count = = 0 then
(13)       Visiual.Add(qToObj);
(14)     else
(15)       if(Occlusion(q, Obj, ObsSet)) then
(16)         Visiual.Add(Obj);
(17)       end if
(18)     end if
(19)   end for
(20) end for
(21) Return Visiual;
(22) END
```

Algorithm 5: kNN algorithm.

random point set was generated within this range. These random point sets were the original point sets that constituted each convex hull. In order to reduce the amount of calculation, it was necessary to further prune the original point set that constituted each convex hull. After obtaining the coordinates of the point set that constituted each convex hull model, the convex hull generation algorithm was used to obtain the outermost surface of each convex hull. The outermost points obtained by the outermost surface were the surface point sets constituting the convex hull, and these surface point sets could represent the convex hull model. The data generation software was used to generate random three-dimensional objects in the space of $400 \times 400 \times 400$, and the data-processed method mentioned above was performed on each three-dimensional object to obtain the corresponding internal convex hull model.

6.2. Analysis of Results. The visual k-nearest neighbor query algorithm proposed in this paper used the GJK collision detection algorithm to determine the occlusion relationships between objects and obstacles for the first time when the query was performed, and then, the line-surface relationship was used to determine the visibility of the objects. For comparative experiment, only the line-surface relationship judgment for visibility was used. In the experiment, it was not only for the line-surface relationship judgment but also the method of the combination of the GJK judgment and line-surface relationship judgment proposed in this paper. In order to describe the two methods better, the line-surface relationship detection method alone was named "line-surface" judgment, and the method of GJK collision detection combined with line-surface relationship detection proposed in this paper was named "GJK" judgment. In addition, R* tree was constructed in the 3D obstacle space to compare with our proposed algorithm, which is called "3D R*-Tree". The experiment would be conducted from two aspects: the size of the data set and the value of the k. The experimental results were analyzed as follows.

The effect of data set size was analyzed on query performance, and the variable data set sizes were set to 50, 100, 150, . . ., and 500, respectively. The data sets were distributed uniformly, and the two algorithms were executed in the same experimental environment. Each experiment was run 5 times under each data set and $k = 20$, and the average value was used as the effective running time for comparison. The run times of the results were shown in Table 1.

These data were plotted into a line chart for analysis, as shown in Figure 14. It could be seen from the figure that as the data set increased, the query time of the algorithms increased responsively, but the query time of "GJK" judgment was always less than the query time of "line-surface" and "3D R*-Tree". This was because the larger amount of data, the number of spatial objects that the algorithms needed to judge whether they were visible was also increased, the "GJK" judgment algorithm proposed in this paper reduced the number of "line-surface" judgments, so the query performance was better. It also could be seen that as the data set increased, the growth rate of the "GJK" judgment would become flat. The "line-surface" judgment and the "3D R*-Tree" had no such trend. This was because as the data set increased, the greater the data density, the greater the possibility that the objects were filtered out farther from the query point, because the objects that were farther away, the fewer objects that were visible completely. So, there were fewer data objects relatively for obstacle distance calculation.

Table 1: Run times under different data sizes.

Data size	GJK	Line surface	3D R*-tree
50	0.092	0.101	0.082
100	0.137	0.315	0.233
150	0.181	0.511	0.382
200	0.232	0.647	0.492
250	0.485	1.147	0.951
300	1.172	2.021	1.821
350	1.702	3.003	2.512
400	2.123	3.847	3.346
450	2.721	4.575	4.012
500	2.954	5.557	4.554

Table 2: Run times under different k values.

k value	GJK	Line surface	3D R*-tree
5	2.022	5.033	3.232
10	2.302	5.103	3.502
15	2.533	5.223	3.833
20	2.754	5.557	4.554
25	2.862	5.688	5.462
30	2.965	5.959	6.365
35	3.246	6.236	6.746
40	3.421	6.588	7.421
45	4.045	6.827	7.935
50	4.962	7.176	8.462

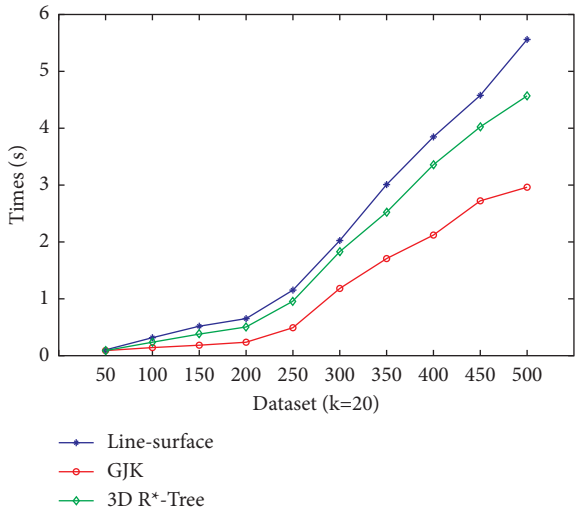

Figure 14: The effect of the data size on the performance of the visual kNN query.

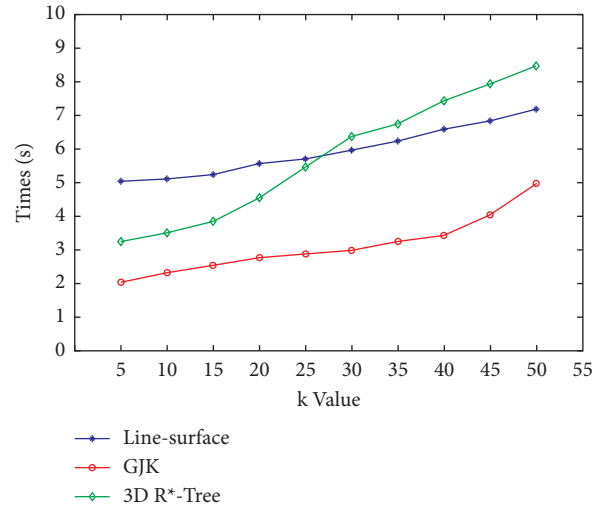

Figure 15: The effect of the k value on the performance of the visual kNN query.

The other two methods needed to calculate all the data points, so they did not have such a trend. The "3D R*-Tree" was constructed without overlapping parent nodes, and there was no multipath query problem, so the query speed was faster than the "line-surface" query. It could be concluded from the experiments that the "GJK" judgment algorithm proposed in this paper had a better performance.

We analyzed the influence of the k value on query performance and set the value of k to 5, 10, 15, . . ., 45, and 50, respectively, and the data set size remained unchanged at 500. The data sets were distributed uniformly, and the two algorithms were executed in the same experimental environment. A statistical method was used to support the experimental results. Each experiment was run 5 times under the value of each k, and the average value was used as the effective running time for comparison. The run times of the results were shown in Table 2.

The comparison of the experimental results of the algorithms was shown in Figure 15. As the value of k increased, all polylines showed an upward trend, but the "GJK" judgment polyline was always lower than the other two algorithms. By filtering objects, the number of objects involved in the calculation would be greatly reduced, which played an important role in the efficiency of the algorithm. The growth rate of the "3D R*-Tree" was the largest. With the increase of query objects, the solution of multiway query could not improve the efficiency of the query effectively. The growth rate of the "GJK" judgment and "line-surface" was similar, but the "GJK" collision detection algorithm was more efficient, so its runtime is shorter than the "line-surface" judgment. The above situation showed that with the continuous increase of the k value, the "GJK" judgment algorithm had better query performance.

7. Conclusion

The spatial query in the obstacle space has received widespread attention as it is closer to the real world. Current studies on the visual query are focused on two-dimensional space mostly and are less involved in three-dimensional space. The objects with complex shapes in the three-dimensional space become a new challenge to the visibility query. Aiming at this problem, a visibility detection method of the three-dimensional object in obstacle space based on the convex hull model is proposed. And, by this method, the visual k nearest neighbor query is carried out. The study of this paper is concluded as follows.

Firstly, related definitions and theorems of the three-dimensional convex hull model are given, and the method to detect the visibility of this model is proposed. The GJK collision detection algorithm is used to determine whether there is a collision between the convex hulls. If a collision occurs, the line-surface relationship is used to further judge whether it is visible. Secondly, the related algorithms of visibility detection based on the convex hull model are proposed. Thirdly, the shortest distance algorithm between the query point and the three-dimensional convex hull and the obstacle occlusion judgment algorithm are given. Finally, the visual k-nearest neighbor algorithm based on the convex hull model is proposed based on the above algorithms.

The visual k-nearest neighbor query algorithm based on the convex hull model proposed in the paper is constructed in an experimental environment. The experimental comparison of two visual k-nearest neighbor query experiments by comparing the setting of the data set size and the size of the k value proves that the k-nearest neighbor query algorithm proposed in this paper has a better query performance.

The study of this paper has extensive application value in the process of industrialization. In the automotive industry, the application of autonomous driving technology can enable cars to avoid obstacles and find the best driving path during autonomous driving. In the medical field, it is used for mechanical surgery to achieve precise tumor resection. In the military field, precision strikes and radar detection of fighter jets can be achieved.

In this paper, the visibility detection method of three-dimensional object is proposed in obstacle space based on the convex hull model to improve the accuracy of the visual query of objects in three-dimensional space. However, in the study of visual query, multiple obstacles that collide with each other are combined into a "combined obstacle," which is also represented by a convex hull. In real life, the combined obstacles formed by the collection of spatial obstacles may be concave. In this situation, the accuracy of the three-dimensional space will be reduced when the visibility is detected. In view of this problem, in-depth thinking and discussion can be carried out in future research. In addition, in the three-dimensional space, the optimal path and the prediction of moving obstacles based on the convex hull model are also the next issues that we will focus on.

Data Availability

Spatial data generator is an open spatial database generation tool. Users can download from the Internet and generate data sets according to their needs.

Conflicts of Interest

The author declares that there are no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation under Grant no. 61972334, 2019 Hebei University Higher Education Science and Technology Research Youth Fund Project under Grant no. QN2019044, and Qinhuangdao science and Technology Bureau, Research and Development of Science and Technology in Qinhuangdao under Grant no. 201902A028.

References

[1] Q. Zhao, S. Zhang, Y. Fan, and A. Hao, "Research on interactive digital model of human organs and virtual surgery," *Scientia Sinica*, vol. 46, no. 12, pp. 1769–1773, 2016.

[2] K. Shao, "The design and implementation of virtual battlefield simulation system," *Electronic Journal publishing of Master*, vol. 3, no. 31, pp. 1–90, 2017.

[3] B. Hurley, "Making VR a NASA reality," *NASA Tech Briefs*, vol. 41, no. 1, pp. 16-17, 2017.

[4] Y. Liu and D. Kong, "Continuous visible query for three-dimensional objects in spatial databases," *Mathematical Problems in Engineering*, vol. 201612 pages, 2016.

[5] S. Alipour, M. Ghodsi, U. Güdükbay, and M. Golkari, "Approximation algorithms for visibility computation and testing over a terrain," *Applied Geomatics*, vol. 9, no. 1, pp. 1–7, 2017.

[6] D. Z. Chen and H. Wang, "Weak visibility queries of line segments in simple polygons," *Computational Geometry*, vol. 48, no. 6, pp. 443–452, 2015.

[7] A. Zarei and M. Ghodsi, "Query point visibility computation in polygons with holes," *Computational Geometry*, vol. 39, no. 2, pp. 78–90, 2008.

[8] M. Ghodsi, A. Maheshwari, M. Nouri-Baygi, J. R. Sack, and H. Zarrabi-Zadeh, "α -Visibility," *Computational Geometry*, vol. 47, no. 3, pp. 435–446, 2014.

[9] V. Polishchuk, E. Arkin, A. Efrat et al., "Shortest path to a segment and quickest visibility queries," *Journal of Computational Geometry*, vol. 7, no. 2, pp. 77–100, 2016.

[10] H. Wang, "Quickest visibility queries in polygonal domains," *Discrete & Computational Geometry*, vol. 62, no. 2, pp. 374–432, 2019.

[11] L. Shou, Z. Huang, and K. Tan, "HDoV-Tree: The Structure, the Storage, the speed," in *Proceedings of the 19th International Conference on Data Engineering*, pp. 557–568, Bangalore, India, March 2003.

[12] M. Kofler, M. Gervautz, and M. Gruber, "R-trees for organizing and visualizing 3D GIS databases," *The Journal of Visualization and Computer Animation*, vol. 11, no. 3, pp. 129–143, 2000.

[13] J. Zhang, D. Papadias, and K. Mouratidis, "Spatial queries in the presence of obstacles," in *Proceedings of the 9th International Conference on Extending Database Technology*, pp. 366–384, Heraklion, Crete, Greece, March 2004.

[14] Y. Gao, B. Zheng, G. Chen, Q. Li, K. C. K. Lee, and W. C. Lee, "Visible reverse k-nearest neighbor query processing in spatial databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1314–1327, 2009, https://doi.org/10.1109/TKDE.2009.113.

[15] X. Yu, Yu Gu, T. Zhang, and G. Yu, "A method for reverse k-nearest-neighbor queries in obstructed spaces," *Chinese Journal of Computers*, vol. 34, no. 10, pp. 1917–1925, 2011.

[16] Yu Gu, X. Yu, and Y. U. Ge, "Method for continuous reverse k-nearest neighbor queries in obstructed spatial database," *Journal of Software*, vol. 5, no. 8, pp. 1806–1816, 2014.

[17] L. Zhang, L. Liu, H. Xiaohong, and L. Song, "Voronoi-based group reverse k nearest neighbor query in obstructed space," *Journal of Computer Research and Development*, vol. 54, no. 4, pp. 861–871, 2017.

[18] L. I. Song, Y. Shao, S. Ning, and L. Tan, "Performance of a newly isolated salt-tolerant yeast strain Pichia occidentalis G1 for degrading and detoxifying azo dyes," *Bioresource Technology*, vol. 233, no. 12, pp. 21–29, 2017.

[19] Z. Yu, Y. Liu, X. Yu, and K. Q. Pu, "Scalable distributed processing of K nearest neighbor queries over moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1383–1396, 2015.

[20] J. Han, K. Chen, Z. Ding, and H. Cao, "An efficient location reporting and indexing framework for urban road moving objects," *Distributed and Parallel Databases*, vol. 32, no. 2, pp. 271–311, 2014.

[21] H. J. Cho, S. K. Choe, and T. S. Chung, "A distributed approach to continuous monitoring of constrained k-nearest neighbor queries in road networks," *Mobile Information Systems*, vol. 8, no. 2, 126 pages, Article ID 303585, 2012.

[22] D. Vorona, A. Kipf, T. Neumann, and A. Kemper, "DeepSPACE: approximate geospatial query processing with deep learning," in *Proceedings of The 27th Acm Sigspatial International Conference on Advances in Geographic Information Systems*, pp. 500–503, Chicago, IL, USA, November 2019.

[23] J. Wan, B. Tang, Y. He, and S. Li, "A continuous k-nearest neighbor query method for moving data in obstructed spaces," *Journal of Harbin University of Science and Technology*, vol. 23, no. 03, pp. 48–54, 2018.

[24] B. Li, C. Zhang, and D. Li, "A DSP-topk query optimization algorithm supporting indoor obstacle space," *Journal of Computer Research and Development*, vol. 54, no. 3, pp. 557–569, 2017.

[25] X. Yu, X. Tang, B. Ye, B. Song, and X. Zhou, "Obstacle Space Modeling and Moving-Window RRT for Manipulator Motion planning," in *Proceedings of the 2016 IEEE international conference on information and automation*, pp. 539–544, Ningbo, China, August 2016.

[26] X. Luo, K. Chen, G. Pang, L. Shou, and G. Chen, "Visible nearest neighbor search for objects moving on consecutive trajectories," in *Proceedings of the 2017 IEEE International Symposium On Parallel And Distributed Processing With Applications And 2017 Ieee International Conference On Ubiquitous Computing And Communications*, pp. 1296–1303, Guangzhou, China, December 2017.

[27] A. Arman, M. Ali, F. Choudhury, and K. Abdullah, "VizQ: A System for Scalable Processing of Visibility Queries in 3D Spatial Databases," in *Proceedings of the 2017 ACM on conference on information and knowledge management*, pp. 2447–2450, Singapore, November 2017.

[28] C. Haider, A. Arman, M. Ali, and F. M. Choudhury, "Continuous maximum visibility query for a moving target," in *Proceedings of the 27th Australasian Database Conference on Database Theory and Applications*, pp. 82–94, Sydney, NSW, Australis, September 2016.

[29] I. Rabban, M. Ali, M. Cheema, and Hashem, "The maximum visibility facility selection query in spatial databases," in *Proceedings of the 27th ACM Sigspatial International Conference on Advances in Geographic Information Systems*, pp. 149–158, Chicago, IL, USA, November 2019.

[30] Q. Meng, H. L. Wang, W. Xu, and Y. Chen, "Numerical homogenization study on the effects of columnar jointed structure on the mechanical properties of rock mass," *International Journal of Rock Mechanics and Mining Sciences*, vol. 124, Article ID 104127, 2019.

[31] Q. X. Meng, H. L. Wang, W. Y. Xu, M. Cai, J. Xu, and Q. Zhang, "Multiscale strength reduction method for heterogeneous slope using hierarchical fem/dem modeling," *Computers and Geotechnics*, vol. 115, Article ID 103164, 2019.

[32] Y. J. Cao, W. Q. Shen, J. F. Shao, and W. Wang, "Numerical homogenization of elastic properties and plastic yield stress of rock-like materials with voids and inclusions at same scale," *European Journal of Mechanics - A: Solids*, vol. 81, Article ID 103958, 2020.

[33] X. S. Shi, J. Nie, J. Zhao, and Y. Gao, "A homogenization equation for the small strain stiffness of gap-graded granular materials," *Computers and Geotechnics*, vol. 121, Article ID 103440, 2020.

[34] Q. Xiong, X. Wang, and A. P. Jivkov, "A 3D multi-phase meso-scale model for modelling coupling of damage and transport properties in concrete," *Cement and Concrete Composites*, vol. 109, Article ID 103545, 2020.

[35] Y. T. Feng and Y. Tan, "On Minkowski difference-based contact detection in discrete/discontinuous modelling of convex polygons/polyhedra," *Engineering Computations*, vol. 37, no. 1, pp. 54–72, 2019.

[36] Q. Meng, H. Wang, M. Cai, W. Xu, X. Zhuang, and T. Rabczuk, "Three-dimensional mesoscale computational modeling of soil-rock mixtures with concave particles," *Engineering Geology*, vol. 277, Article ID 105802, 2020.