U.P.B. Sci. Bull., Series C, Vol. 85, Iss. 1, 2023 ISSN 2286-3540

A TREE STRUCTURE FOR STORAGE, ACCESS AND COLLISION DETECTION OF DISKS

S. Sohrabi[1], A. Moradinejad[2], A. Asgharian[3]

This paper presents a new and efficient tree data structure for sorting and collision detection of disks in 2D based on a new tree-based data structure, called hexatree, which is introduced for the first time in this paper. The data structure relies on mapping the objects to a space 1 dimension higher and then doing an appropriate range query in that range which is 3D range queries with circular cones of fixed direction and opening angle. Once the tree is constructed in $O(n \log n)$ time, disk collision detection queries can be answered in time $O(n^{0.61})$. The proposed algorithm, in principle, is a dynamic structure; that is, it's a structure that can be modified once it's built. The proposed algorithm can be used for designing CAD softwares and other issues that require searching for a disk among a multitude of 2D disks.

Keywords: data structure, collision detection of disks, CAD environments

1. Introduction

In modeling process of data sets, there is a huge amount of data in geography and many other disciplines [14, 15] that leads to regular geometric shapes such as disks and polygons. Since this resources will grow exponentially, the storing and analysing data become even more ubiquitous [16]. Consequently we need to find efficient data structures for navigating through these data sets.

In computer science, a data structure is a set of data values, the relationships among them, and the functions or operations that can be applied to the data [18, 3]. For example, in many cases, we need to know all the disks in a large set that satisfy a certain relation with a given disk. This is known as a disk query problem: given a set of data disks S, a query disk Q, and one relation R, a disk query is to retrieve all the disks $C \in S$, such that the relation $C\,R\,Q$ satisfies. The most common and widely studied problem in disk query, is the collision detection problem where the relation R refers to the collision relation [21]. As it turns out, searching a disk among a multitude of two-dimensional disks is required and it is clear that to add or search a disk,

[1] Department of Mathematics, Faculty of Science, Urmia University, Urmia 57561-51818, Iran, e-mail: `s.sohrabi@urmia.ac.ir` (Corresponding author)

[2] Faculty of Engineering, Urmia University, Urmia, Iran

[3] Department of Computer Engineering, Faculty of Electrical and Computer Engineering, Urmia University, Urmia 57561-51818, Iran, e-mail: `a.asgharian@urmia.ac.ir`

one should not explore all the disks. On the other hand, in a structure, the most important issue is the flexibility of the environment to work with objects like disks, lines and arcs. So the environment and the used objects should not be supposed rigid, since during the modeling process the objects may be changed permanently. For solving this problem, the tree-based methods were further developed. The most important reason for choosing trees is their high screening ability.

The aim of this paper is to develop a novel tree data structure to storage, access and collision detection of a set of disks by using a new tree-based data structure. This data structure can also be used for designing CAD softwares and other issues that require searching for a disk among a multitude of two-dimensional disks. The existing data structures such as sphere tree and octree mostly dose not have the necessary efficiency. So by increasing the degree of nodes, the expense for searching the disks which intersects the query disk is very economic. In order to sort the disks, we introduce an inventive dynamic tree that has shown good efficiency in inserting nodes and searching. The main contributions and results of the paper are as follows:

- We introduce a new tree-based data structure for disk collision detection.
- Using our data structure, we introduce a fast method for disk collision queries.
- We also introduced a method for updating our data structure i.e. methods for adding or removing disks.

The remainder of the paper is organized as follows. In section 2, the related works have been reviewed. Section 3 contains our proposed data structure and algorithm for the collision detection of disks. In section 4, we analyzed the effectiveness of our algorithm against existing methods. Finally, in section 5, summarizing the main results we conclude the paper.

2. **Related work**

Trees are most common hierarchical data structures used to solve geometrical problems [3, 9]. Trees are hierarchical arrangement of items in which the items are represented as being “above”, “below”, or “at the same level as” one another. In the geometric problems, the hierarchical arrangement usually is constructed by recursively subdividing space and can represent spatial phenomena at a variety of geographic scales with successively larger scales represented on successively lower tiers of the hierarchy. They have been used in GIScience and computational geometry as an efficient means of storing spatial data, as a fast way of retrieving that data and as a multiscale representation [9]. In order to construct a hierarchical tree, firstly a fixed object is considered. Then, the enclosing rectangular cube or sphere or other types of geometric shapes is used to divide the space occupied by the object. For example, in AABB [20] and OBB [17] methods, a rectangular cube containing the entire object is first found. Then the center point of the object volume is determined and then the rectangular cube will have two children, each of which will include half of the object that extends from the vertex of the original rectangle to the center of the object. This creates two children for the original rectangular cube and this process continues recursively on children until it reaches the basic element of the object, such as a triangle. A typical example for such a tree-based geometric data structure is the quadtree [10, 19]. Quadtrees were introduced for the first time by Finkel and Bentley [4] as a suitable data structure for answering queries about sets of points in multi-dimensional space. A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees are the two-dimensional analog of octrees and are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. These subdivided regions may be square or rectangular, or may have arbitrary shapes. The other related works on object query made use of interval trees [11], segment trees [2, 8, 13], range trees [7], k-d trees [1], interval sorting [6], sphere tree [5, 12] and etc.

In the case of collision detection problems, because the output size of a collision detection query may be $O(n)$, the worst case running time for any query algorithm is $O(n)$; however, some algorithms are output sensitive and their running time is specified with both the number of objects n and the size of the query result (or the number or reported objects) m. As an example, the interval trees which are used for checking the overlapping intervals have a query time of $O(\log n + m)$ and an initial creation (or preprocessing) time of $O(n \log n)$, while limiting memory consumption to $O(n)$. The segment tree which is also a binary tree for arrangement of intervals, have a query time of $O(\log n + m)$, and $O(n^2)$ space complexity. The range trees are another type of binary trees that are used for arrangement of intervals, but the query time is $O(\log^2 n + m)$, and their total construction and storage time is $O(n \log n)$ [2]. Because segment trees can be generalized to 2-dimensional space, one can use an $O(\log n + m)$ algorithm for finding the overlapping rectangles. Also, k-d tree data structure [1] can be used in 2-dimensions to create such an algorithm for rectangles. However, none of the above mentioned data structures and algorithms do not give an efficient method for querying disks overlaps. A naive approach to find all overlapping disks with a given query disk, is to check the query disk with all he disks which requires $O(n)$ time. In this paper, we give a data structure and an output sensitive algorithm to find the all overlapping disks in time $O(n^{\log_6 3} + m)$ in which m is the number of overlapping disks with the query disk. In other words, our algorithm is an output sensitive algorithm and its running time is better than $O(n)$ approach when the size of output is less than $O(n)$. To be able to answer queries in $O(n^{\log_6 3} + m)$ time, our algorithm uses a tree-based data structure called hexatree. The hexatree contains all the disks and is constructed incrementally by adding disks. Inserting each disk in the hexatree take $O(\log n)$ time, so the time to build the hexatree is $O(n \log n)$, which is regarded as preprocessing time.

3. Proposed method for collision detection of disks

In this section we describe the structure and construction of proposed tree structure for sorting and querying 2D disks. This idea can also be extended for other regular shapes such as polygons with their special conditions (for example, when circulation is not permitted).

3.1. The structure and construction

In the proposed method, we will firstly convert each disk in the collection S, into a unique 3D point in the conventional 3D space XYR, where the X and Y axes are used to represent the coordinates of the center and the R-axis is used to show the radius of the disk.
Now, we consider two disks $A, B \in S$ as follows:

$$\text{disk } A : (x - x_A)^2 + (y - y_A)^2 = r_A^2,$$
$$\text{disk } B : (x - x_B)^2 + (y - y_B)^2 = r_B^2,$$

Suppose that (x_A, y_A, r_A) and (x_B, y_B, r_B) are the corresponding 3D points for the disks A and B, respectively. Then, the trivial condition for their collision is

$$d_{AB} \leq r_A + r_B, \tag{1}$$

where

$$d_{AB} = \left((x_A - x_B)^2 + (y_A - y_B)^2\right)^{1/2}. \tag{2}$$

On the other hand, we know that

$$\left((x - x_0)^2 + (y - y_0)^2\right)^{1/2} \leq z - z_0, \tag{3}$$

denotes the upper half of a simple standard cone where the angle at the apex of the cone is 90 degrees, i.e., if a plane passes through the central axis of the cone, the cross-section is a right triangle where the lateral lines of the triangle have a slope of ± 1.
Now we consider a query disk $Q : (x - x_Q)^2 + (y - y_Q)^2 = r_Q^2$, with a non-negative radius and map it into the 3D point $(x_Q, y_Q, -r_Q)$ such that the vertex of the corresponding test (query) cone coincides on it (see Fig. 1).
With the above assumptions, a point P is on or inside the test cone, if and only if the relation

$$\left((x_P - x_Q)^2 + (y_P - y_Q)^2\right)^{1/2} \leq r_P + r_Q,$$

holds, which is the same codition for collision of two disks. Therefore, the corresponding disk for each point on or inside the test cone intersects with the query disk.
It needs to be explained here that if we use octree [6], then as shown in Fig. 2, two nodes (or sometimes depending on the position of the current node, three child nodes) of eight nodes can be ignored certainly, which are not very desirable. In Fig. 2, the child nodes of the current node that are not scrolled during the search process, are shown in pink and the blue disk is the same

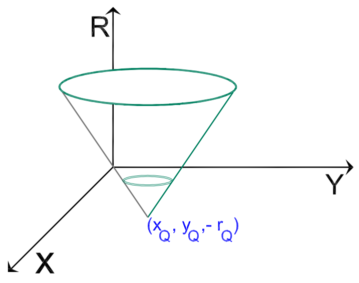

FIGURE 1. The test cone corresponding with the query disk.

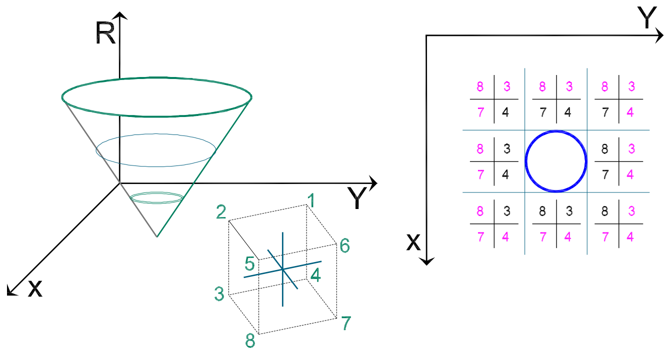

FIGURE 2. The principles of search process in octrees.

cross-section disk that the position of the node is evaluated whit respect to it. If the node is located inside the disk, all of its children nodes are traversed.

At this stage we need to sort the converted points properly and effectively. For this purpose, we propose a new six-nodes tree, which we call it as "hexatree" (see Fig. 3). The main reason for choosing this new tree is that we need an space partitioning that matches with the standard cone, especially with the slope of standard cone. Also in tree traversing, as few nodes as possible should be visited among the nodes according to their position in the converted environment (compared to the test cone) to continue the search path. Therefore, the best partitioning is obtained by considering the position of each point as the center of a cube. By connecting this point with the vertices of each side of the cube, we obtain six pyramids, as shown in Fig. 4. Since we are only dealing with four pages connected to the center of the cube, in practice, we consider the base of each pyramid to be infinite. In this way, the 3D space is divided into the six completely separated partitions without a

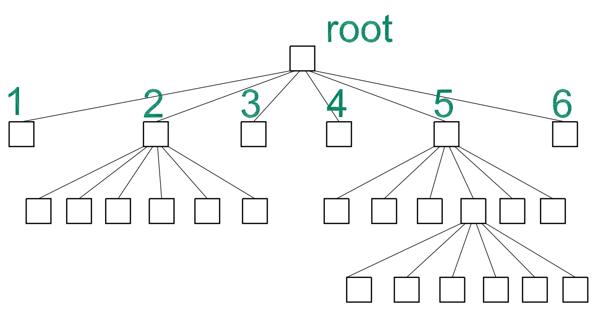

Figure 3. A sample of the haxatree.

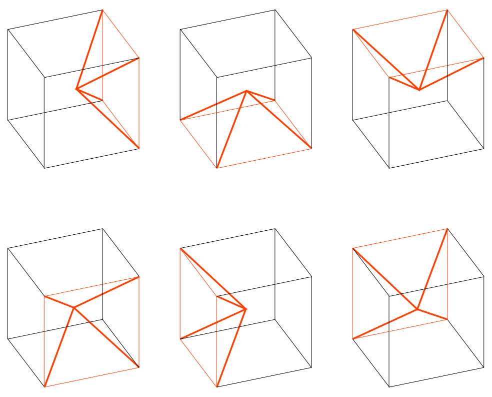

Figure 4. Six pyramids in a cube.

bulky intersection. We remark that the above mentioned principles for octrees are also used for search process in the new tree.

3.2. The traversal for inserting new node (new disk)

If the tree is empty, we make it as the first node and consider six empty nodes for it. Depending on which of the six pyramids of the root node is located in, the next node is transferred to the corresponding node among the empty children of the root node, and then six empty children are also considered for this node. The process is the same for the next nodes: It starts from the root node and then finds the first step of the path and goes to the appropriate child node. If it is empty, it is added there. Otherwise, according to its

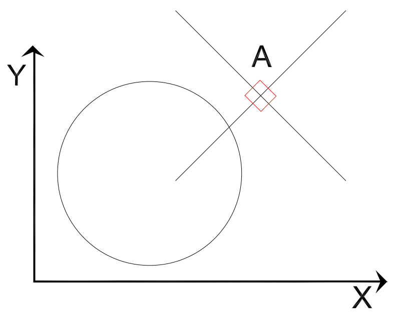

FIGURE 5. The intersection of the test cone at the height of the node A.

position relative to the middle node, it goes to the appropriate child node. This process continues until the node is added. The cost of operation of this part is logarithmic and it is done quickly.
In order to determine whether a point is located in the pyramid or not, we define the vector $\vec{d} = \vec{P} - \vec{C}$, where C is the coordinates of the center of the cube and P is the desired point for which we are looking to find the appropriate child node index. By the above assumptions, we know that if

$$\langle d, n_i \rangle \leq 0, \quad i = 1, 2, 3, 4, \tag{4}$$

where $n_i,\ i = 1, 2, 3, 4$ denote the normal vector of the four side planes of the pyramid, then the point is located inside the pyramid. We must care that the direction of all normal vectors should be out of the pyramid. If even one of the inner products in (4) be positive, then the point is not located inside the desired pyramid.

3.3. The construction algorithm

In the case of the hexatree algorithm, we need an important and simple function which examines the test cone. To improve the search performance of the hexatree, we use a simple geometric principle. If we consider the cross-section of the test cone at the height of the current node A, Fig. 5 occurs. Here we present an argument that increases the effective ratio of the intended tree. For this purpose, we first introduce the "cross-section disk" as follows:

Definition 3.1. *For a current node A, the cross-section disk is a cross-sectional area of the test cone at the height of the piont A, where the normal vector of its plane is aligned with the R-axis.*

Having the test cone and the converted point related to a disk, to know the position of the point relative to the cone, we need the concept of cross-section disk. Using this disk, it is easy to identify that which children of the node

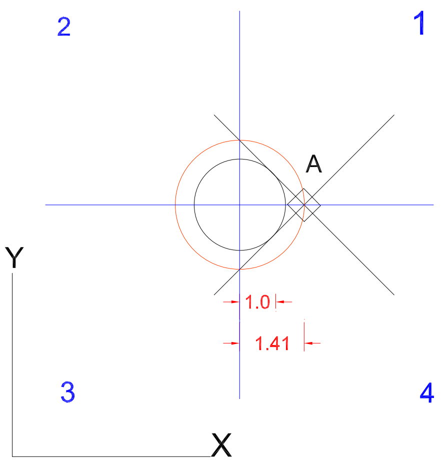

FIGURE 6. The light-red disk is the criterion disk at the node A.

should not be followed during the search process. To improve the search performance, we consider a hypothetical disk where its center is the center of the cross-section disk and its radius is the $\sqrt{2}$ times the radius of the same disk. We call it as the "criterion disk" (see Fig. 6). An important feature of the criterion disk is that in the search process, one can easily ignore traversing 3 out of 6 sub-trees of a node which leads to a higher performance search. If the node A is located in the first quarter (as is shown in the Fig. 6), we can easily ignore the 3 child nodes in searching process. For the other nodes, the same event happens, only the indexes of the explored nodes are different.
From the above discussion, the effective ratio will be near to 3/6 and in cases where the average of the radii is near to the radius of a node, it will get a better answer. Now, we are ready to present the algorithm of adding nodes. The adding algorithm requires a simple function to determine the index of the child in which the new node should be inserted depending on the geometric position of the input argument. We call it by Child_Index in Algorithm 3.1. The AddNode procedure in Algorithm 3.1 only does a single recursive call on one of its children, so its running time is easily proportional to the height of the tree. Because we expect the height of the tree to be $O(\log_6 n)$, the running time of AddNode is $O(\log n)$.

3.4. The search algorithm

For the search algorithm (Algorithm 3.2), the function TwoConeCollision is used to investigate the collision of two standard cones where the values of p, q, s, k, u, v, w, z are obtained according to the geometric position of the present node and the intersection of the cone with a plane with normal vector

Algorithm 3.1 AddNode

```
procedure AddNode($t \in nodes; x, y, r \in \mathbb{R}$)
    if $t \neq NULL$ then
        $j$ =Child_Index($t \to x, t \to y, t \to r, x, y, r$)
        AddNode($t \to child(j), x, y, r$)
    else
        $new(t)$
        $t \to x = x$
        $t \to y = y$
        $t \to r = r$
        for $i = 1$ to 6 do
            $t \to child(i) = NULL$
        end for
    end if
end procedure
```

$(0, 0, 1)$ at the height of the intended node. We remark that the location of the point in each quarter of the coordinates of the standard disk is a decisive issue and in searching process the maximum performance is obtained when the points are out of the criterion disk.

A simple analysis shows, however, that for a reasonably balanced tree with inner nodes of degree p for which a search continues at q children, about $O\left(n^{(\log q/\log p)}\right)$ nodes are visited for $q \geq 2$. Therefor, by considering the above algorithm, we observe that the search process is performed recursively for $q = 3$ child nodes out of $p = 6$ nodes. Therefore, the recursive relation for the time complexity of the Search procedure is as follows:

$$T(n) = 3T(n/6) + O(1),$$

Which means time complexity of Search is $O\left(n^{(\log 3/\log 6)}\right) \approx O(n^{0.61})$.

4. Discussion and comparison

As we know, there are no algorithms or data structures for the disk collision query problem. A naive approach for this problem is to test the given query disk with all the disks which requires time $O(n)$. However, our proposed algorithm can do such queries is $O(n^{0.61})$ expected time. Instead, our algorithm require to build a hexatree for the given set of disks to be able to answer the queries. It means that, our algorithm requires $O(n \log n)$ preprocessing time. Therefore, it is efficient for the applications that demand a lot of collision queries. Also, our algorithm is useful in dynamic application in which the disk may be removed or inserted, because we can remove or insert a node in the hexatree in $O(\log n)$ time. Therefor, the effectiveness of our algorithm in comparison with naive approach is clear. However, one may use some of the existing data structures to prone the search space, in give efficient algorithm. The most similar data structure for hexatree is octree. It is not possible to

Algorithm 3.2 Search

```
procedure SEARCH($t \in nodes; x, y, r \in \mathbb{R}$)
    if $t \neq NULL$ then
        if TWOCONECOLLISION($t \to x, t \to y, \sqrt{2}(r + rmax), x, y, r$) then
            if TWOCONECOLLISION($t \to x, t \to y, t \to r, x, y, r$) then
                $result = t$
            end if
            SEARCH($t \to child(1 \to 6), x, y, r$)
        else
            SEARCH($t \to child(TopChild\_Index), x, y, r$)
            if $t \to x \leq x$ then
                if $t \to y \leq y$ then
                    SEARCH($t \to child(p), x, y, r$)
                    SEARCH($t \to child(q), x, y, r$)
                else
                    SEARCH($t \to child(s), x, y, r$)
                    SEARCH($t \to child(k), x, y, r$)
                end if
            else
                if $t \to y \leq y$ then
                    SEARCH($t \to child(u), x, y, r$)
                    SEARCH($t \to child(v), x, y, r$)
                else
                    SEARCH($t \to child(w), x, y, r$)
                    SEARCH($t \to child(z), x, y, r$)
                end if
            end if
        end if
    end if
end procedure
```

acquire a asymptotic bound on the time complexity of such a method which use octree instead of hexatree to find disk collisions. Therefore, we can not compare them in terms of time complexity. Thus, we have conducted a experimental comparison between the octree and the hexatree (our algorithm) for preprocess and search times (by milliseconds). The result of the experiments is given in Table 1, where n is the total number of disks. The disks are generated with random radii and positions. As we see in Table 1, the search time for the presented tree in this paper is less than the octree, while both trees are made from the same disk set. Also, an implementation of the proposed algorithm has been shown in Fig. 7, for collision detection of a query disk (in red) with a set of 10000 disks (in black) with random radii in $[10, 40]$ and centers in $[1, 1000] \times [1, 1000]$. The total number of visited disks (in green) is 551, while the number of answer disks (in yellow) is 100. We observe that only 5% of disks has been visited.

TABLE 1. The comparison between the octree and the hexatree in cunstruction and searching disks.

	Preprocess Time (ms)		Search Time (ms)	
n	hexatree	octree	hexatree	octree
10000	5.25	3.5	0	0
100000	52	26	0.25	2.5
1000000	1207.5	466.75	2.5	20
5000000	5432.5	1919.5	16.25	100.5
10000000	10496.6	8086.667	28.2	522.6

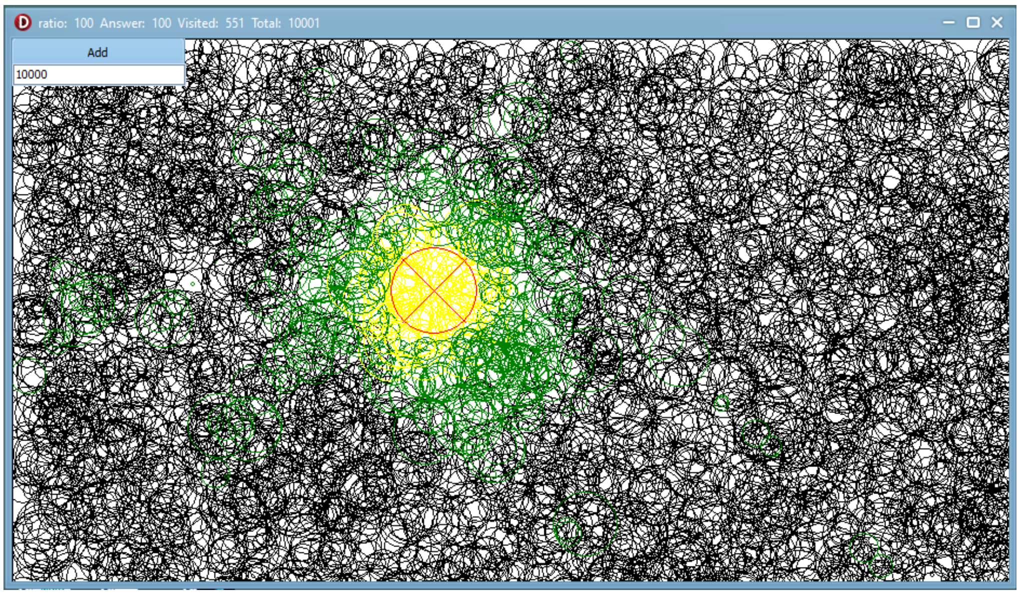

FIGURE 7. An implementation of the proposed method for disk query.

5. Concluding remarks

In this paper, a new method based on the new 6-degree tree structure called hexatree has been presented for effective collision detection of disks. Because of our space partitioning method, each internal node in a hexatree has 6 children. After building a hexatree for a set of disks, we can answer disk collision queries very fast. An advantage of hexatree in comparison with the other rigid methods such as sphere tree is that it is dynamic, i.e. we can insert or remove disks dynamically. The idea presented in this paper can be extended for querying other regular geometric shapes such as polygons and will be reported in future works. A limitation of our method is that it works only on 2D disks. Therefore, extending the algorithm for 3D space can be addressed in future works. Also, giving a worst-case upper bound for the running time of our search algorithm is another issue for our future works.

REFERENCES

[1] *J. L. Bentley*, Multidimensional binary search trees used for associative searching, Communications of the ACM. **18(9)**(1975), 509-517.

[2] *M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf*, Computational Geometry: Algorithms and Applications, 3rd edition, Springer-Verlag, Berlin, 2008.

[3] *T.H. Cormen, C.E. Leiserson, R.L. Rivest, Clifford Stein*, Introduction to Algorithms, 3rd edition, The MIT Press, Cambridge, 2009.

[4] *R.A. Finkel, J.L. Bentley*, Quad trees; a data structure for retrieval on composite keys, Acta Informatica, **4(1)**(1974), 1-9.

[5] *P. Hubbard*, Approximating polyhedra with spheres for time-critical collision detection, ACM , 1996. Transactions on Graphics, 15, 3, 179-210.

[6] *B. J. Lucchesi, D. D. Egbert, and F. C. Harris*, A parallel linear octree collision detection algorithm, International Journal of Computer Applications, **21 (4)**(2014), 230–243.

[7] *G. S. Lueker*, A data structure for orthogonal range queries, 19th Annual Symposium on Foundations of Computer Science (1978), 21-28

[8] *D.P. Mehta, S. Sahni*, Handbook of Data Structures and Applications, 2nd Edition, Chapman and Hall/CRC, 2018.

[9] *A. Moore*, The disk tree-a hierarchical structure for efficient storage, access and multi-scale representation of spatial data, Presented at SIRC 2002 – The 14th Annual Colloquium of the Spatial Information Research Centre University of Otago, Dunedin, New Zealand, December 3-5th 2002.

[10] *P. Ottoson, and H. Hauska*, Ellipsoidal quadtrees for indexing of global geographical data. International Journal of Geographical Information Science, **16,3**, 213-226.

[11] *A. Pal and M. Pal*, Interval tree and its applications, dvanced Modeling and Optimization, **11 (3)**(2009), 211-226.

[12] *I J. Palmer and R L. Grimsdale*, Collision detection for animation using sphere-trees, Computer Graphics, 1995. Forum, 14, 2, 105-116.

[13] *F.P. Preparata, M.I. Shamos*, Computational Geometry: An Introduction, 3rd edition, Springer-Verlag, Berlin, 1990.

[14] *P. Rigaux , M. Scholl and A. Voisard*, Spatial Databases with Application to GIS. Morgan Kauffmann, 2002.

[15] *H. Samet*, Applications of Spatial Data Structures. Addison Wesley, 1990.

[16] *H. Samet*, The Design and Analysis of Spatial Data Structures. Addison Wesley, 1990.

[17] *H. Songhua, Y. Lizhen*, Optimization of collision detection algorithm based on OBB, 2010 International Conference on Measuring Technology and Mechatronics Automation, **doi: 10.1109/ICMTMA.2010.460**

[18] *P. Wegner, E.D. Reilly*, Encyclopedia of Computer Science. Chichester, UK: John Wiley and Sons, 2003.

[19] *M. Worboys*, GIS-A Computing Perspective, Taylor and Francis, 1995.

[20] *W. Xiao-rong, W. Meng, L. Chun-gui*, Research on collision detection algorithm based on AABB, 2009 Fifth International Conference on Natural Computation, **doi: 10.1109/ICNC.2009.196**

[21] *Y. Zhao, Q. Nie, L. Xu and B. Li*, A Survey of Continuous Collision Detection, 2020 2nd International Conference on Information Technology and Computer Application (ITCA), 2020, 252-257.