International Journal of Intelligent Information and Management Science

Volume 2, Issue 5, June 2013

http://www.hknccp.org

Contents

HK.NCCP

International Journal of Intelligent Information and Management Science
ISSN: 2307-0692 Volume 2, Issue 5, June 2013

A Multi-task Pathfinding Method Based on CBR and Kd-tree

Yan Li, Sen Liang, Lanming Su, Jiacheng He
Key Lab. In Machine Learning and Computational Intelligence
College of Mathematics and Computer Science, Hebei University
Baoding, China

Abstract: Pathfinding is a typical task in many computer games, and its performance will affect the quality of game AI. In order to enhance the efficiency of multi-task pathfinding, case-based reasoning has been introduced in traditional A* algorithm, called the CBMT method. The method needs to select representative paths which can cover the whole map to build a compact case base, which is difficult in large maps. Besides, repeatedly searching for similar cases for each pathfinding task would be a time consuming process. To address these problems, we provide a kd-tree case storage structure and case retrieval mechanical in the CBMT method. The pre-stored cases (previously found paths) are generated randomly and incrementally. The original flat storage structure of the cases is changed into the kd-tree structure. Since the searching space can be reduced by branch pruning in case retrieval, the pathfinding efficiency has been improved obviously, and the number of searched nodes is also reduced.

Keywords: Multi-task Pathfinding; Case-based Reasoning; Kd-tree; Nearest-neighbors Search Algorithm; A*

1. Introduction

Pathfinding is an important and typical task in many computer games, especially in real-time strategy games. An excellent pathfinding algorithm will surely increase the satisfaction degree of users. A* [1] is one of the most often used heuristic search algorithms, which can find the optimal solution when there does exist a shortest path. Although A* is the typical pathfinding algorithm, it suffers from high time and space complexity. There are many improvements about A* in the literature, such as HPA* (Hierarchical Path-Finding A*) [2], LPA* (Lifelong Planning A*) [3-4], KNN LRTA* (k-Nearest Neighbor classification algorithm and Learning Real-Time A*) [5], among others. HPA* is a typical hierarchical pathfinding algorithm based on the concept of abstract graph. It firstly abstracts a large map into several linked local clusters. In each cluster, several representative nodes are selected and then linked by A* to generate the abstract graph. At the global level, clusters are traversed in a single big step. At the local level, the optimal distances for crossing each cluster are pre-computed and stored. It can greatly reduce the problem complexity in path-finding on grid-based maps. However, HPA* is merely suitable for static environments. When the obstacle information is changed during the gameplay, the found path needs to be replanning and updated in real time. LPA* was proposed in 2004, which can fast update the shortest path in dynamic environments by storing information from previous pathfinding results. Its first search is similar to A* but the subsequent searches are potential faster because it reuses the previous search information.
However, it was reported to be very sensitive to the position of the changed nodes, and the pathfinding task (i.e., the pair of start node and destination node) should be fixed beforehand. In some related applications such as computer games, multi-task pathfinding is more often required for game players because they need to continuously move from one place to another or the multi-players need to find their paths simultaneously. Therefore, the start and destination nodes changed from time to time. If we start from the scratch for each task, the strict limit of response time for computer games cannot be satisfied. In this situation, Case-Based Reasoning (CBR)[6-7] has been used in the multi-task pathfinding, and finds each path by reusing the previously found similar paths. CBR mimic the behavior of human being which retrieves and reuses past experience to solve new problems. It has been widely applied in many different real problems [8-10], including in real-time games. In 2009, Vadim Bulitko, et al proposed KNN LRTA*, which combines K-Nearest Neighbor algorithm [11] with LRTA* [12]. The LRTA* algorithm is a real time heuristic search algorithm, which gets the optimal path by expanding its front state and updating its heuristic in the unknown environment and changing environment. In the offline phase of KNN LRTA*, the randomly found paths by A* are compressed into a set of subgoals and stored as cases in the case base. When a new task comes, the algorithm will query the case base for the most similar previously solved case and

uses its subgoals to solve the new problem. It should be noted that, computer games require high efficiency for pathfinding, while CBR needs to retrieve the case base for similar cases when each new task occurs which is very time-consuming. In [13], a case-based multi-task pathfinding algorithm called CBMT is developed, which is an incremental memory-based method. This algorithm can obviously reduce the number of searched nodes as more and more found paths (cases) are accumulated. The query task generated randomly can be different at each time. Through computing the distance between the new task and the stored cases, the most similar one can be reused and adapted as the result of the new task. This newly found path is then also stored as a success case and this completes one CBR cycle. Obviously, the CBMT algorithm can reduce the online search time at the cost of using more memory. However, there are still some disadvantages: Firstly, it is difficult to get the representative paths in large map. Secondly, it will take a lot of time to searching for similar paths in the case base. In this paper, we improve the storage structure of the case base and the case retrieval mechanical by using the kd-tree structure. The cases can still be stored incrementally and the tasks can be dynamic. First, many pre-stored paths are generated randomly to cover the search space as much as possible. Then the original flat storage structure of the cases is replaced by using kd-tree structure, and the KNN searching mechanical is also changed. The experimental results show that, the proposed new method can improve the speed of searching similar cases obviously and thereby enhancing the efficiency of the pathfinding task.

The remainder of this paper is organized as follows. Section 2 explains the idea of kd-tree and the application of kd-tree in pathfinding. In section 3, we describe the proposed multi-task pathfinding algorithm in detail. Section 4 shows the experimental analysis and results. Finally, we provide the conclusion and the future work in Section 5.

2. Kd-tree for Pathfinding and the Case Retrieval Algorithm

Kd-tree (k-dimensional tree) [14] is a data structure which can partition the high dimensional search space with hyper-rectangles. It is mainly used to quickly retrieve the required information in large volume of data since it can reduce the search space by branch pruning during information query. Kd-tree has been used in some problems which have large search spaces such as 3D Point Pattern Matching [16], Real-time Huge Terrain Visualization[17], and Satellite RCS Prediction[18]. In fact, kd-tree is an extension of binary search tree to k-dimensional data, and it is different from binary tree because of a discriminator in each layer to decide the trend of the branch.

2.1. Case Representation and Kd-tree Forming Process

In this paper, the kd-tree is used to structure the pre-stored paths to build a case base to facilitate fast pathfinding. First, we should determine the dimension of the kd-tree, i.e., the k value. Generally, we can represent a path by its start node and destination node, and their coordinates can be used to form a case. Then we use two pairs of x-coordinate and y-coordinate to represent both a case and task. Therefore, k is equal to 4 in the proposed kd-tree structure. Each case is a four-dimensional data point Pi (xi, yi, zi, ki), i∈[1, n]. The process to form such a kd-tree is described as follows.

(1) Calculate the median of every component of all the data point. Then we take the median as the root of the tree.

(2) Determine which component to be selected as the discriminator using the formula d=Lmod4, where L is the serial number of the kd-tree' hierarchy. When the value of d equals to 0, 1, 2, 3 respectively, the components of xi, yi, zi, and ki will be selected correspondingly as the discriminator in the tree construction.

(3) When the component value in the current node Pi is less than or equal to the discriminator, then we put the data point Pi into the left subtree, otherwise, put Pi into the right subtree.

(4) Keep loop until all the points are inserted into the kd-tree.

Figure 1 is a kd-tree built in the way mentioned above. Each tree point in the kd-tree is a four-tuple (startX, startY, endX, endY), which represents a task with start state (startX, startY) and goal state (endX, endY). The search space is divided in the order of startX, startY, endX, and endY, and the loop repeats until the kd-tree is built successfully.

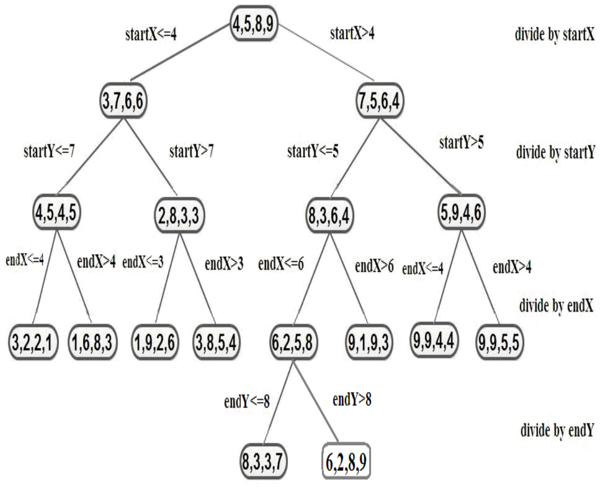


Figure 1. The Kd-tree Built by 17 Points

2.2. Case Retrieval Based on Nearest Neighbor Principle

Kd-tree is mainly used to reduce the search space, so as to improve the search efficiency. After building a tree, the next key issue is case retrieval. Here we use the nearest neighbor (NN) principle [15] to find the most similar case for each task. With the nearest-neighbor search algorithm in kd-tree, we do not need to search all the cases one by one by eliminating some subtrees. For example, when a new task comes with the start state (8, 4) and the goal state (4, 9), and the task will be represented by a kd-tree point (8, 4, 4, 9). The search process is shown in figure 2. Firstly, it searches in the right subtree of the root as the startX = 8 which is greater than the discriminator 4 of the root; then on the second level, the left subtree of the next node is searched as startY(=4) is less than the discriminator 5; at the third level, it searches the left subtree as its value of 4 for endX is less than the discriminator 6. Finally, on the fourth level, it searches the right subtree as its endY = 9 which is greater than the discriminator 8. This process ends until it attains a leaf node, e.g., (6, 2, 5, 9) in Fig. 1. This leaf node is considered as the most similar to the task (8, 4, 4, 9) and will be reused and adapted to complete the new task.

However, if we check more carefully, it is not difficult to find that the found leaf node (6, 2, 5, 9) is not the most similar case stored in the kd-tree. In fact, a back tracking process should be included after the leaf node is obtained. Firstly, we give a definition of case similarity simply based on Manhattan distance between two points P1 and P2 noted as d(P1, P2). The leaf node in the kd-tree which has the minimum distance to the given task P (x, y, z, k) is considered as the most similar case. That is, P* is found as the retrieved case when Mini d(Pi, P) = d(P*, P) and P* is a leaf node in the kd-tree.

Secondly, the nearest-neighbor algorithm is used to pick out the most similar case from the case base. Based on the computed similarities of the new task and the stored cases in the kd-tree, one leaf node can be finally obtained when the search loop goes to the bottom level of the tree. This leaf node is not necessarily the most similar case. We should continue search the right case by backtracking. As shown in figure 2, the dotted lines with arrows are used to describe the backtracking process. The found leaf node is (6, 2, 5, 9) which can be regarded as a candidate of similar case. The following steps are:

(1) Set the current leaf node as the nearest neighbor node, and then calculate the Manhattan distance d0 between this node and the new task.

(2) A hypersphere H centered at the task (8, 4, 4, 9) is defined with radius d0.

(3) The parent node (6, 2, 5, 8) of the leaf node is revisited in such a way: If the hypersphere H intersects with that induced by the parent node, we will search the left subtree of the parent node. Otherwise, we will eliminate the left subtree so as to enhance the efficiency of searching. Obviously, in the example showed in Fig 2, the H intersects with the hypersphere induced by the parent node (6, 2, 5, 8). So we keep on searching the left subtree of this parent node.

(4) Update the minimum distance and nearest neighbor node. We will find that the Manhattan distance between the task and the current candidate node (8, 3, 3, 7) is smaller than d0, and then the minimum distance is updated. The current node (8, 3, 3, 7) is set as the nearest neighbor node.

(5)The process continues until the root node is backtracked. In the end, the most similar case in the whole case base can be obtained, i.e., (8, 3, 3, 7) in this example.

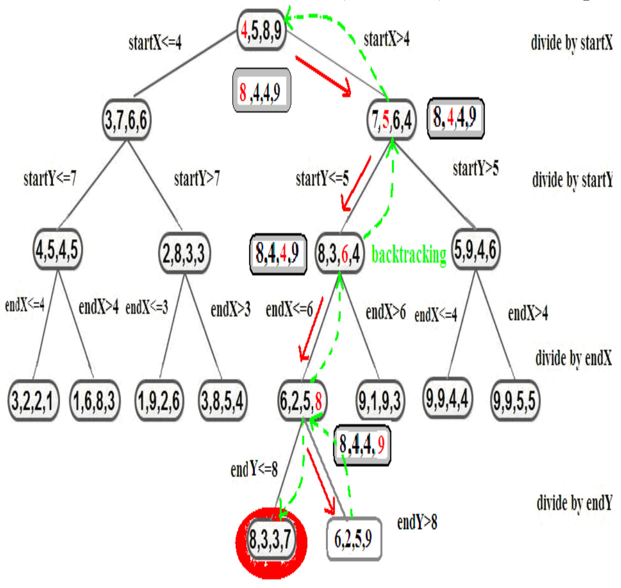


Figure 2. The Kd-tree for Casebase Access

Through the above case retrieval process, it can be seen that the kd-tree based on NN algorithm can guarantee finding the most similar case as well as reducing the search space effectively.

3. Multi-task Pathfinding Based on CBR and Kd-tree

As we previously mentioned, case-based reasoning has been incorporated in A* algorithm for multi-task pathfinding, which is called CBMT algorithm. Figure 3 shows the process. When a new task is coming, it no longer uses A* to find a path from the scratch, but checks the cases in the memory firstly to decide whether to go along the previous stored paths or not. After the similarity computation, if an enough similar case to the new task can be successfully retrieved from the case base, we will make use of this retrieved path to complete the new task. Therefore, the searching time will be reduced to the time of looking up the memory. With the increasing number of stored paths(cases), the CBMT pathfinding algorithm can reduce the price of A* obviously. However, there are still some issues to be addressed. Firstly, the pre-stored paths are manually selected so as to ensure that the paths are representative (i.e., paths can cover the whole map). The case selection method will not practical in large maps. Secondly, since the cases are stored in a flat structure, searching for the most similar path will take up a lot of time with increasing number of cases. This will deteriorate the efficiency of CBMT algorithm.

To address these problems, we propose a new multi-task pathfinding algorithm based on CBR and kd-tree. Firstly, we increase the number of pre-stored paths generated randomly in large maps to cover the whole map as much as possible. Since the new task will also be randomly generated, if the cases cover larger region in the map, the larger probability of the most similar cases can be successfully found. Besides, in order to improve the quality of stored paths, the threshold is set as half of the map's edge length. Only when the length of the randomly generated path is greater than or equal to the threshold, can it be stored in the case base. Otherwise, we drop it and regenerate a new path. After each CBR cycle, the successfully solved task will also be saved in the case base for further use. With the continuous arrival of new tasks, the number of the cases stored in the case base is also increasing. Meanwhile, more comparisons are needed for similarity computation, which is a time-consuming process. Therefore, secondly, we change the original flat structure of the case base to the kd-tree structure. The speed of searching similar case can be improved greatly, and then enhancing the efficiency of the algorithm. Additionally, kd-tree also support dynamic case base, in which the cases are stored incrementally.

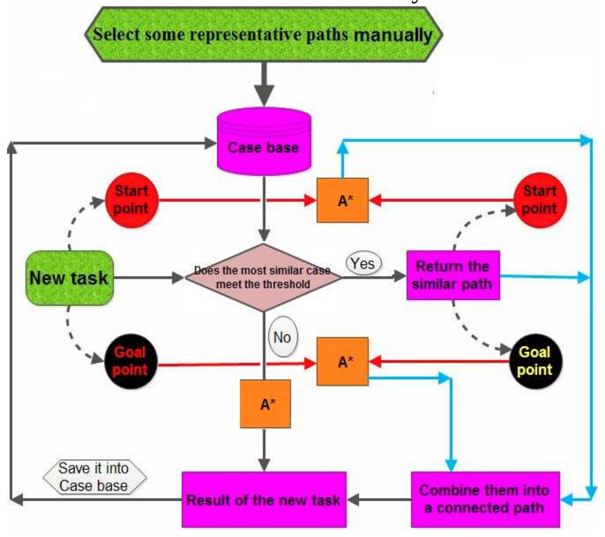


Figure 3. Flow Chart of the CBMT Pathfinding Algorithm

Figure 4 shows the new proposed pathfinding method. Here we firstly define three notations: Vstart, Vgoal and pathi, where Vstart and Vgoal denote the start node and the destination node of the new task, respectively; and pathi is the ith path stored in the memory. The process mainly consists of two phases: offline and online, which is described as follows.

Offline phase: building a case base and stored it as a kd-tree.

Online phase: this phase starts when a new task comes which includes the following steps.

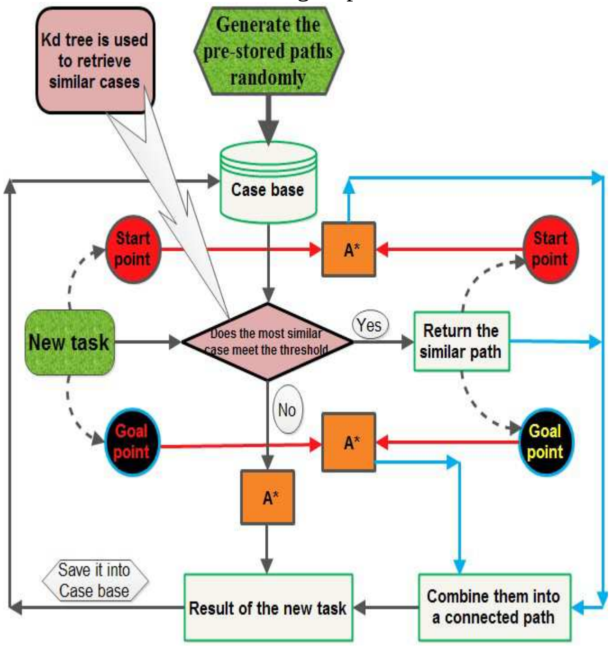

Kd tree is used to retrieve similar cases
Generate the pre-stored paths randomly
Case base
Start point
A*
Start point
New task
Does the most similar case meet the threshold
Yes
Return the similar path
No
Goal point
A*
Goal point
A*
Save it into Case base
Result of the new task
Combine them into a connected path

Figure 4. Flow Chart of the Improved Version of the CBMT Pathfinding Algorithm

Using nearest neighbor search to query the most similar case to the given task in the kd-tree.

Determine whether the retrieved case can be reused according to the distance threshold.

Supposing pathi is one case stored in the kd-tree. It will be considered reused and adapted to solve the new task only when it is retrieved as the most similar case as well as it meets the threshold condition.

Step 1 Calculate the Manhattan distance d1 between Vstart and the start node of pathi and also calculate the Manhattan distance d2 between Vgoal and the destination node of pathi.

Step 2 Compare the summation of d1 and d2 and the set threshold. If the summation is greater than the threshold, we consider that even the most similar case found in the case base is not similar enough to solve the task. In this case, A* will be still used for the pathfinding task. Otherwise, we will reuse the selected pathi.

(3) Reuse and adapt the retrieved case.

Step 1. Use A* to search for a path from Vstart to the start point of pathi.

Step 2. Use A* to search for the other path from Vgoal to the goal point of pathi.

Step 3. Combine the found two paths together with pathi to form a complete conncected path from the Vstart to Vgoal. In other words, the new task is solved by our proposed method.

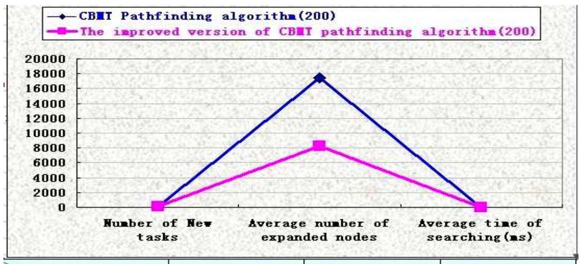
Obviously, we only run A* in two small segments rather than the whole path. The main part of the path is generated by reading from the cache rather than searching by A*.

Therefore, our method can avoid the deficiency of A* by using the cases stored in the case base as a kd-tree. With the increasing number of the stored paths, the available information will be accumulated quickly and facilitate the fast problem-solving. This method supports dynamic and incremental data storage and fast NN searching. The experimental results will demonstrate its potential advantage in the reduction of the searching complexity.

4. Experimental Results

Our experiment is carried out in a map of 500×500 pixels. At the beginning, we store 200 randomly generated paths as the case in advance. In order to guarantee the quality of the generated paths, we require the length is larger than the half of the map's edge length. If the length of the path generated randomly is greater than or equal to the threshold, we save it to the case base. Otherwise, we drop it and regenerate a new path. In addition, we take account of the length of the new task. If the length of the new task is too short, there is no need to implement search process in this proposed method. A* algorithm can meet the demand of the task with low price.

We test our improved version of CBMT pathfinding algorithm by 100 new tasks. As we can see from Figure 6, the average number of the expanded nodes of CBMT pathfinding algorithm is 17482, however, the average number of the expanded nodes of the improved version of CBMT pathfinding algorithm is 8206. The average searching time of CBMT pathfinding algorithm is 4.68 millisecond, and that of the improved version of CBMT pathfinding algorithm is 2.96 millisecond. This shows the improved version of CBMT pathfinding algorithm is more efficient with a less space cost. In other word, using the storage structure of kd-tree, the time of similarity computation has been reduced obviously. Therefore, the speed of searching the similar case can be improved greatly, and then enhancing the efficiency of the algorithm.



	Number of New tasks	Average number of expanded nodes	Average time of searching (ms)
CBMT Pathfinding algorithm (200)	100	17482	4. 68
The improved version of CBMT pathfinding algorithm (200)	100	8206	2. 96

Figure 5. Comparisons of the Improved Algorithm with the CBMT Pathfinding Algorithm

5. Conclusions and Future Work

In this paper, case-based reasoning and kd-tree are used in multi-task pathfinding, which can reduce the search space effectively. With the increasing number of the stored paths (cases), most parts of the new task is no longer to complete by A*, but by reusing the pre-stored cases. Therefore, the proposed method can get the path more easily and quickly. However, with the increasing number of the cases, the space complexity will still increase quickly. In the future work, we will further trim the kd-tree so as to make sure all the cases in the case base are representative.

Acknowledgements

This work is supported by NSFC (No. 61170040), the natural science foundation of Hebei Province (F2010000323; F2012201023), and the youth natural science research plan foundation of Hebei University (No. 2010Q023).

References

[1] P. Hart, N. Nilsson, B. Raphael. A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics, 4(2): 100-107, 1968.

[2] A. Botea, M.Muller, and J. Schaeffer. Near optimal hierarchical pathfinding, Journal of Game Development, 1(1): 7-28, 2004.

[3] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A*, Artificial Intelligence, 155(1-2): 93-146, 2004.

[4] Y. Lu, X. Huo and O. Asiotras, et al. Incremental multi-scale search algorithm for dynamic path planning with low worst-case complexity, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 99:1-15, 2011.

[5] V. Bulitko, Y. Bjornsson, R. Lawrence. Case-based subgoaling in real-time heuristic search for video game pathfinding, Journal of Artificial Intelligence Research 39, 269-300, 2010.

[6] A. Aamodt, E. Plaza. Case-based reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. IOS Press, 7(1): 39-59, 1994.

[7] P. Cunningham. A taxonomy of similarity mechanisms for case-based reasoning, IEEE Transactions on Knowledge and Data Engineering, 21(11): 1532-1543, 2009.

[8] H. Li, J. Sun. Gaussian case-based reasoning for business failure prediction with empirical data in China, Information Sciences, 179(1-2): 89-108, 2009.

[9] D. McSherry. Conversational case-based reasoning in medical decision making, Artificial Intelligence in Medicine, 52(2): 59-66, 2011.

[10] Y. Du, W. Wen, F. Cao, and M. Ji. A case-based reasoning approach for land use change prediction, Expert Systems with Applications, 37(8): 5745-5750, 2010.

[11] Hastie, T., Tibshirani, R. Discriminant adaptive nearest neighbor classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(6): 607-615, June 1996.

[12] Korf, R.E. Real-time heuristic search. Artificial Intelligence 42(2-3):189-211, 1990.

[13] Y. Li, L. Su, Q. He, Case-based multi-task pathfinding algorithm, Proceedings of the 2012 International Conference on Machine Learning and Cybernetics, Xi'an, 15-17July, 2012.

[14] Bentley, J.L. Multidimensional binary search trees used for associative searching. Corn. of ACM, vol.18: 509-517, 1975.

[15] Moore, A. Efficient memory-based learning for robot control. Ph.D. thesis, University of Cambridge, 1991.

[16] B. Li, H. Holstein. Using k-d trees for robust 3D point pattern matching. In 4th International Conference on 3D Digital Imaging and Modeling (3DIM 2003), 6-10 October 2003, Banff, Canada. IEEE Computer Society, pages 95-102, 2003.

[17] D. Yao, J. He, and W. Liu. A real-time huge terrain visualization algorithm based on kd-tree. Science Technology and Engineering. 12 (2): 338-341, 2012.

[18] Ge Zhao, Jun Zhang and Jiemin Hu. A modified satellite RCS prediction method based on ray tracing combined with kd-tree descriptions. Telecommunication Engineering. 52(5): 712-715, 2012.