3-Dimensional Shortest Path Searching and Chasing in the Volumes

Gene Eu Jan[1] Tong-Ying Juang[1] Su Chien-Min[2] Wan-Rone Liou[2]
[1]*Department of Computer Science*
National Taipei University, Taipei, Taiwan
gejan@mail.ntpu.edu.tw juang@mail.ntpu.edu.tw
[2]*Department of Electrical Engineering*
National Taiwan Ocean University, Keelung, Taiwan
cmsu@mail.ntou.edu.tw wrliou@asic3.ee.ntou.edu.tw

Abstract*-This paper presents the 3D shortest path searching algorithms in the volumes with O(N) of time and memory space complexities based on the nuclear fission chain reactions scheme, where N is the number of voxels in the grid space. Furthermore, the proposed algorithm is extended to the 3D shortest path searching for multiple pairs based on the concept of the aircraft domain to avoid collision and chasing algorithm in the volumes with the time complexity of O(qN) and $O(N^2/a)$, respectively, where q is the number of aircraft and a is the relative speed ratio of chaser to target. The concept of these algorithms can be applied to GIS (Geographic Information System), search and interception for aircrafts, and cruise missile interception system.*

1. Introduction

In the recent years, the shortest path searching problem has been widely studied in the literatures. The shortest path means to find a path between a *source* (*start*) position and a *destination* (*end*, *goal*, *target*) position with the minimum *distance* or *time* (In this article, we prefer the time instead of distance since each aircraft might have different speed). There are two different approaches among all path-searching algorithms: *graph* (*vector*)-based and *grid* (*raster*)-based approaches. The graph-based approach consists of two phases. The first phase is associated with the construction of a graph representing relations between free spaces (non-obstacle nodes). Once the graph is obtained, in the second phase, the optimal path referred to a certain criterion (shortest path, minimum time, etc.) has to be found. One of the well-known graph search methods is Dijkstra's algorithm in which the problem is obstacles dependent. Another approach, the grid-based approach that is obstacles independent, can be used to find an optimal path on the cell map or volume. In the earlier 1960's, Lee [1] presented the shortest-route algorithm using 2D planar cells, that can be applied to VLSI and PCB design, maze games, and searching on raster map problem [2][3]. The key for the algorithm is widely accepted because the concept of the algorithm is simple and ease to implement. The computation of the algorithm only involves insertion and deletion from a linked list and the access for each cell, although the memory buffer may be large to store the required raster data. Since the technology of the semiconductor rapidly increases memory capacity, this problem is no longer to be emphasized. But, the result of Lee's algorithm is confined to a rectilinear path, thus its applications is limited. To improve the above problem, the higher geometry maze routing scheme is introduced in 2D planar space [4].

Recently, we are interested in the 3D shortest path problems in the volumes as well. By contrast, 2D raster is forming with *pixel*. The *voxel* represents the element of the volumes. The volume's data structure can be divided into two classes: vector structure and volume structure. In vector structure, the shortest distance between two points is the Euclidean length of the path. In volume structure, the shortest path between two voxels has the least sum of distances from the source voxel to the destination voxel through the consecutive neighboring voxels. Earlier works on the shortest path-searching problem were centered on 2D image space. Following the advancement of learning, these researches spread in the volumes. Such as, Li et al. [6] presented a software system incorporates motion-planning algorithms in Robotics into the latest development of network-based virtual reality standards such as VRML 2.0. Li et al. [7] describe a planner capable of generating humanoid motions in 3D space on stair-like terrains by taking the human foot length and personal preference into consideration. The 3D shortest path algorithm in the volumes was presented using the *nuclear fission chain reaction* method with an extra data structure that has 26-directional expansion [5]. The algorithm was developed from an idea that stemmed from the process involving neutrons hitting neighboring atoms with each neighboring atom releasing new neutrons. The chain reaction spreads into the entire space until every atom has been split exactly once. This process is terminated when all of the atoms have released their neutrons. Our method proposed in this paper for the 3D shortest path algorithm in the volumes with the time and memory space complexities of *O*(N). The

algorithm will be applied to search the 3D shortest path and collision avoidance for multiple pairs and 3D real time chasing system.

The rest of the paper is organized as follows. The proposed 3D shortest path algorithm in the volumes is introduced in section 2. The 3D shortest path algorithm and collision avoidance scheme for multiple pairs, 3D real time chasing system and their corresponding examples are illustrated in section 3. Finally, the conclusion of the study is presented in section 4.

2. The 3D shortest path algorithm

The technology of GIS has various applications in aviation and military, for example, the autopilot system [8] is applied to the aircrafts and vessels. In this paper, the proposed algorithm can be applied to not only PCB and VLSI designs to reduce the length of the path but also GIS application. In the 2D image space or the volumes, the proposed algorithm has a great advantage by simplifying the computations of the free space and obstacles to find the shortest path. Other implementations are path planning of traffic navigation, search and rescue for aircrafts and vessels, and interception in the GIS with the auxiliary of GPS.

In the following subsections, the volumes' data structure is first presented, the 3D shortest path algorithm in the volumes and its detailed method is described.

2.1. The data structure and definitions

Comparing with graph-based approach, grid-based approach has some advantages such as the data is easy to process and analysis [9]. In this paper, the proposed method is based on the grid-based approach to find the 3D shortest path. Using the grid-based approach, the arbitrary shape of obstacles in the volumes can be easily rasterized. The proposed searching scheme runs faster if the obstacles are fairly complicated in the volumes since the searching time for the non-obstacle space is reduced. The volumes data structure is constructed from the concept of the 2D pixel plane. In an $I \times J \times K$ volumes, any voxel $C_{i,j,k}$ has four parameters for voxel storage, $O_{i,j,k}$, $Vis_{i,j,k}$, $IL_{i,j,k}$ and $AT_{i,j,k}$, where $0 \le i \le I-1$, $0 \le j \le J-1$ and $0 \le k \le K-1$. Each voxel has at most 26 neighbors, and the distances of neighboring voxels are $1, \sqrt{2}$ and $\sqrt{3}$ as shown in Figure 1(a), (b), and (c), respectively. The first parameter *O* (*Obstacle*) distinguishes whether a voxel is an obstacle, the Boolean value is *TRUE*, or in the free workspace where the value is *FALSE*. The second parameter *Vis* (*Visited*) distinguishes whether the voxel has visited all its neighbors or not and its initial Boolean value is *FALSE*. The third parameter *IL* distinguishes whether the voxel has been inserted into temporary list *TL* or not and its initial Boolean value is *FALSE*. The fourth parameter *AT* (*Time of Arrival*) stores the time needed to travel from the source voxel to the current voxel and its initial value is infinity.

(a) Distance = 1 (b) Distance = $\sqrt{2}$ (c) Distance = $\sqrt{3}$

Figure 1. Illustration of neighboring voxels in the volumes.

2.2. The 3D shortest path algorithm in the volumes

The 3D shortest path algorithm in the volumes presented in this paper is a substantial improvement of Jan's previous algorithm by reducing the time complexity to *O*(*N*) with a combination of the *nuclear fission chain reaction* method with an extra data structure [10]. The 3D shortest path algorithm in the volumes is introduced as follows.

Algorithm 1: The 3D shortest path algorithm

Step 1: Initialization

For each voxel, $C_{i,j,k}$ in an $I \times J \times K$ volume, the initial $O_{i,j,k}$ =*TRUE* if voxel $C_{i,j,k}$ is in the obstacle or *FALSE* if it is in the free workspace. $Vis_{i,j,k}$ =*FALSE* and $AT_{i,j,k} = \infty$ for all voxels, where $0 \le i \le I-1$, $0 \le j \le J-1$ and $0 \le k \le K-1$. The initial value of the *index* is 0.

Step 1.1: Input the coordinates of the source voxel *S*, if $O_{i,j,k}$ =*FALSE* then update $AT_{i,j,k} = 0$ otherwise return the error message "the source voxel that you spotted is in the obstacle".

Step 1.2: Input the coordinates of the destination voxel *T*, if $O_{i,j,k}$ = *TRUE* then return the error message "the destination voxel that you spotted is in the obstacle".

Step 1.3: Insert the source voxel *S* into the LL_{index}.

Step 2: Update the $AT_{i,j,k}$ values of $C_{i,j,k}$' neighboring voxels $C_{i',j',k'}$.

Step 2.1: Remove the indices of the first voxel $C_{i,j,k}$ from the front end of the LL_{index}.

Step 2.2: For each voxel, $C_{i,j,k}$, in the LL_{index}, update the $AT_{i,j,k}$ values of its neighboring voxels $C_{i',j',k'}$ and set $Vis_{i,j,k}$ to *TRUE*.

Step 2.2.1:

Case 1: $|i-i'|+|j-j'|+|k-k'|=1$, set

$$NewAT_{i',j',k'} = AT_{i,j,k} + 1$$

Case 2: $|i-i'|+|j-j'|+|k-k'|=2$, set

$$NewAT_{i',j',k'} = AT_{i,j,k} + \sqrt{2}$$

Case 3: $|i-i'|+|j-j'|+|k-k'|=3$, set

$$NewAT_{i',j',k'} = AT_{i,j,k} + \sqrt{3}$$

Step 2.2.2: If $NewAT_{i',j',k'} < AT_{i',j',k'}$ then

$$AT_{i',j',k'} = NewAT_{i',j',k'}$$

Step 2.2.3: Insert the indices of $C_{i',j',k'}$ into the *TL* if $IL_{i',j',k'}$ is *FALSE* and then update $IL_{i',j',k'}$ to *TRUE*. ($IL_{i',j',k'}$ is the insertion flag)

Step 2.3: If LL_{index} is not empty, then repeat step 2.1

Step 2.4:

Step 2.4.1: Remove the indices of $C_{i',j',k'}$ from *TL*. and insert it into $LL\lfloor AT_{i,j,k} \rfloor_{\mathrm{mod}\,4}$.

Step 2.4.2: If *TL* is not empty, repeat step 2.4.1

Step 3: Iterations.

Step 3.1: $index=(index+1)_{\mathrm{mod}\,2}$

Step 3.2: If LL_{index} is not empty, then repeat step 2.

Step 4: Backtracking

Step 4.1: If the $AT_{i,j,k}$ value of the destination voxel is infinity, return the error message "there is no path between the source voxel and the destination voxel". Otherwise, backtrack the shortest path from the destination voxel.

Step 4.2: Selecting $C_{i',j',k'}$ of the 26 voxel-connected neighbors with the smallest $AT_{i,j,k}$ value.

Step 4.3: Insert $C_{i',j',k'}$ into path list.

Step 4.4: Repeating the selection of $C_{i',j',k'}$ step by step until the source voxel is reached.

END {The 3D shortest path algorithm in the volumes}

2.3. Performance analysis and illustrative example

The algorithm is applied to the volumes in which the complicated direction movement is simplified by the *nuclear fission chain reaction* scheme. To reduce the repeated computation in the *search wave*, one extra flag is needed to justify the search voxel's status. The algorithm has the same time and space complexities of $O(N)$, where N is the number of voxels in the volumes. One example is shown to demonstrate finding the 3D shortest path in the volumes. In this example, the volumes is divided into 50×50×50 voxels. It is assumed that the red voxel (expressed by *S*) represents the source voxel with (*x*, *y*, *z*)=(1, 31, 1) and the green voxel (expressed by *T*) represents the destination voxel which coordinates is

(a) Front view (b) Top view

Figure 2. Illustration of a 50×50×50 volume.

(0, 25, 27), the result of searching the shortest path is shown in Figures 2, where the russet, black and blue areas represent obstacles, passable areas in the volumes, and the bottom of the volumes, respectively, and the pink path indicates the obtained shortest path. This is convenient for us to inspect how the path avoids hitting the obstacles.

3. Applications and analysis

The 3D shortest path and collision avoidance algorithm for multiple pairs and the 3D real time chasing algorithm with the time complexity of $O(N)$ will be introduced as follows. In addition, the examples of these two algorithms are illustrated in this section.

3.1. The 3D shortest path-searching for multiple pairs in the volumes

The simple way to solve the 3D shortest path-searching problem for multiple pairs in the volumes is to search the shortest path for each pair separately based on the 3D shortest path algorithm in the volumes. However, in the same volumes, if multiple pairs exist simultaneously, it is possible collide each other. In past researches, Colley *et al*. [11] showed that computer simulation of marine traffic flow and collision avoidance. Davis *et al*. [12] presented the computer simulation results for multi-ship encounters. Coenen *et al*. [13] offered a combination of expert system and knowledge base, to prevent collision. The concept of *ship domain* in 2D plane can be extended to the concept of *aircraft domain* in the volumes, such that one can find the shortest paths without collision with each other and nearby obstacles, for multiple pairs. In general, two aircrafts will collide in the volume if two aircrafts are in the same space simultaneously. There are two ways to prevent collision: firstly, change course of aircraft. Secondly, change speed of aircraft. In 2D plane, vessels prevent collision according to the international regulations for collision-avoidance at sea. Every vessel shall at all time keep safe distance in sight of one another to avoid collision, where safe distance is defined by the concept of ship domain. Similarly, the 3D shortest path search with collision avoidance algorithm for multiple pairs in the volumes is based on the concept of aircraft domain to decide safe distance for each other.

3.1.1. The 3D shortest path and collision avoidance algorithm for multiple pairs.
In this subsection, we apply the concepts of aircraft domain and *space-marking* method to this algorithm that is introduced as follows.

Algorithm 2: The 3D shortest path and collision avoidance algorithm for multiple pairs

Step 1: Obtain the shortest path for each pair.
For each pair M_k, the source voxel S_k and destination voxel T_k to the 3D shortest path algorithm in the volumes are indicated. The corresponding shortest path is obtained, where $1 \le k \le$ total number of pairs, respectively.

Step 2: The collision detection.
Simulate all of the paths with aircraft domain and mark their *AD* (aircraft domain) value in the volumes from time to time.

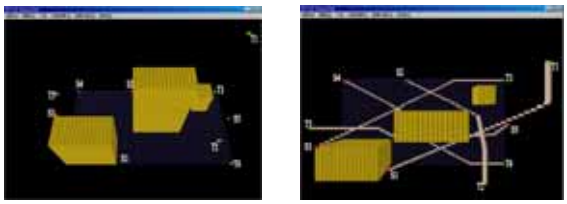
Step 3: The collision avoidance scheme.
If any two pairs have the same *AD* value in the some voxels, the rule of international regulations for collision-avoidance at sea decides which pair needs to change *course* and mark this area as impassable zone for this pair.

Step 3.1: Recompute the path for this give-way pair.
(It is noticed that the recomputed path is no longer the shortest path, it is an optimal path.)

Step 3.2: Simulate the new shortest path.
If the new shortest path collides with another pair, then erase the new shortest path and do the routing with dynamic aircraft domain. Otherwise return "The shortest paths are obtained".

END {The 3D shortest path and collision avoidance algorithm for multiple pairs}

3.1.2. The example and performance analysis.
The example of the 3D shortest path and collision avoidance algorithm for multiple pairs is illustrated in Figure 3. An example of 5-pairs is illustrated in Figure 3(a), where the red voxels (expressed by *S*1, *S*2, *S*3, *S*4, *and S5*) represent the source points and the green voxels (expressed by *T*1, *T*2, *T*3, *T*4, *and T5*) represent the destination points. The top view result of the proposed algorithm is shown in Figure 3(b), where the pink paths represent the desired shortest paths for 5-pairs without collision each other and nearby obstacles. This algorithm is based on the 3D shortest path algorithm in the volumes with the time complexity of $O(N)$. The shortest path algorithm should be called q times for q-pairs in executing the shortest path and collision avoidance algorithm for multiple pairs, its time complexity thus is $O(qN)$.



(a) Initiation (b) Top view

Figure 3. Illustration of the 3D shortest path and collision avoidance for multiple pairs in the volumes.

3.2. Chasing system in the volumes

The proposed algorithm is originally developed for finding optimal routes with collision detection and avoidance in ECDIS (*Electronic Chart Display Information System*) and DEM (*Digital Electronic Maps*). The result of searching for the single-pair shortest path and the time domain collision-free multiple-pairs path was quite satisfied [14]. In general, most of existing path planning algorithms use A* algorithm to search a given moving target. It can be divided into two classes: offline search such as A*, and real-time search such as Real-Time-A* (RTA*), Learning Real-Time-A* (LRTA*) and Dynamic A* (D*) [15]. Among them, some of the moving target search algorithms [16] adopt Manhattan distance along the grid as the initial heuristic value, which is similar to the 2-geometry maze router. The Manhattan distance represents the actual distance if there were no obstacles, but it becomes less accurate as the number of obstacles grows. However, our proposed method used cell map with the higher geometry maze router and time matching scheme to obtain the optimal (least time) path between a searcher and a given moving target [17]. A search and rescue for MIN-MAX and MIN-AVG methods was implemented, and a real time chasing technique was also presented.

In this study, our proposed method is extended from 2D raster plane into the volumes. In the meanwhile, the directional changes of dynamic multiple moving targets are considered as well. Because of the 3D real time chasing algorithm is capable of intercepting the moving targets that the static one cannot. It means that the chasing will be very efficient if the chaser is able to detect the current position of the moving target. Thus, it is very practical to foresee the beforehand path of the target. The 3D real time chasing algorithm is introduced as follows.

3.2.1. The 3D real time chasing algorithm.
It is assumed that the chaser and target have different constant speeds, and the chaser should be faster than the target to catch up. This algorithm involves several steps. First of all, apply the rectilinear forecast that determined the moving target path between last position and current position. After that, next position can be foreseen. Secondly, search the optimal chasing path by the 3D shortest path algorithm in the volumes. Finally, the actual chasing path is determined by the relative speed ratio of chaser to target.

Algorithm 3: The 3D real time chasing algorithm

Step 1: initialization

Step 1.1: Input the initial position of the target $C^{T}_{i(0),j(0),k(0)}$, w=0, z=0.

Step 1.2: Input the initial position of the chaser $C^{S}_{i(0),j(0),k(0)}$

Step 1.3: Input the relative speed ratio of chaser to target, a.

Step 2: Compute the time of arrival between the chaser and the remaining voxels.

Step 2.1: Compute the time of arrival between the chaser and the remaining voxels by the 3D shortest path algorithm in the volumes. In the meanwhile, obtain the present time of arrival $AT^{S(z)}_{T(w)}$ between target and chaser.

Step 3: Foresee the movement of the target.

Step 3.1: Input next moving voxel of the target, $C^{T}_{i(w),j(w),k(w)}$, where w=w+1.

Step 3.2: Foresee the movements of the target linearly.

Step 3.2.1: $C^{Foresee}_{i(0),j(0),k(0)} = C^{T}_{i(w-1),j(w-1),k(w-1)}$, m=0, where m is the parameter of the foreseeing path of the target.

Step 3.2.2:

WHILE $AT^{Foresee}_{i(m),j(m),k(m)} < AT^{S(z)}_{T(w)}$

$C^{Foresee}_{i(m+1),j(m+1),k(m+1)} = C^{Foresee}_{i(m),j(m),k(m)} + C^{T}_{i(w),j(w),k(w)} - C^{T}_{i(w-1),j(w-1),k(w-1)}$

Set the $AT^{Foresee}_{i(m+1),j(m+1),k(m+1)}$ according to the ratio of AT values of $C^{Foresee}_{i(0),j(0),k(0)}$ and $C^{Foresee}_{i(m+1),j(m+1),k(m+1)}$.

If the $O_{i,j,k}$ of $C^{Foresee}_{i(m+1),j(m+1),k(m+1)}$ is *TRUE*, then *BREAK*.

Otherwise m=m+1.

END WHILE

Step 4: Obtain the interception voxel and foresee the beforehand chasing path for chaser.

Step 4.1: Obtain the interception voxel.

FOR $C^{Foresee}_{i(m),j(m),k(m)}$, m=0 TO the target's end voxel of the foreseeing path.

IF $(AT^{S}_{i(m),j(m),k(m)} / AT^{Foresee}_{i(m),j(m),k(m)} = a)$

THEN $C^{Foresee}_{i(m),j(m),k(m)}$ is the foreseeing interception voxel of the target.

Step 4.2: Foresee the beforehand chasing path for chaser.

Starting from the foreseeing interception voxel of the target, step by step, searching the one of 26 neighboring voxels with minimum time of arrival till the starting voxel of chaser is reached. Reversing the beforehand chasing path for chaser, $C^{Beforehand}_{i(0),j(0),k(0)}$, $C^{Beforehand}_{i(1),j(1),k(1)}$,…, $C^{Beforehand}_{i(g),j(g),k(g)}$, $C^{Beforehand}_{i(h),j(h),k(h)}$.

Step 5: Compute the chasing path for chaser.

g=0, where g is the parameter of the beforehand chasing path of chaser.

WHILE $AT^{Beforehand}_{i(g),j(g),k(g)} < AT^{T}_{i(w),j(w),k(w)}$

$C^{S}_{i(z+1),j(z+1),k(z+1)} = C^{Beforehand}_{i(g),j(g),k(g)}$

g=g+1, z=z+1

END WHILE

Step 6: Decide whether the target is caught or not.

If the position of target is not the position of chaser then input next position of target, repeat steps 3 to 6, otherwise, the target is caught.

END {The 3D real time chasing algorithm}

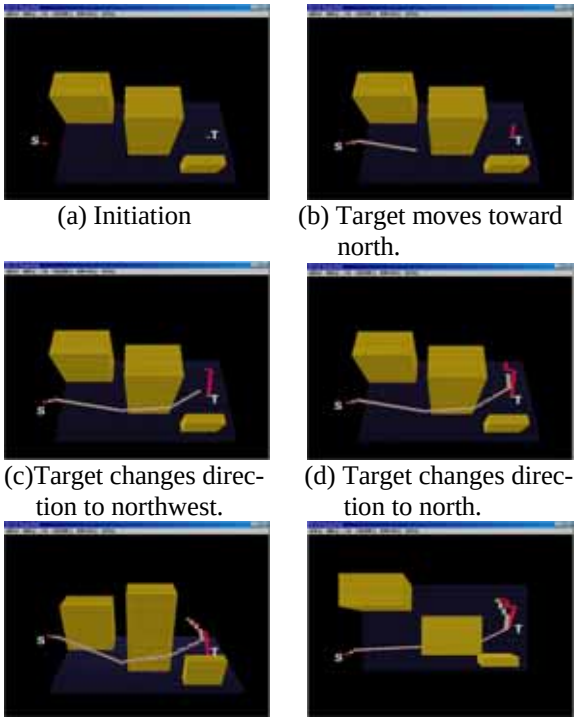
3.2.2. Performance analysis and illustrative example.

In this proposed chasing algorithm, the $T_{Set\text{-}up}$ is defined as the set up time initially for the volume's data structure, $T_{Shortest}$ is the running time of the 3D shortest path algorithm in the volumes, $T_{Foresee}$ is the required time to foresee the movements of the target linearly, and T_{Times} is the number of times of chasing. Since T_{Times} depends on the relative speed ratio a and the different terrains, we only analyze the time complexity of chasing for just once.

$$\begin{aligned} T_{Chasing(1)} &= T_{Times} \times (T_{Set-up} + T_{Shortest} + T_{Foresee}) \\ &\le O(N/a) \times (O(N) + O(N) + O(N)) \\ &= O(N^2/a) \end{aligned}$$

Thus, this algorithm has the time complexity of $O(N^2/a)$.

The illustrative example for the 3D real time chasing is shown in Figures 4(a) through 4(f). In the volume of 50×50×50 voxels, a chaser (expressed by *S*) intends to chase a moving target (expressed by *T*) with the assumed relative speed ratio of 1.5. The red and green voxels represent the chaser and the target, respectively, and their initial positions are shown in Figure 4(a). If the target initially moves toward north, the chaser will find an optimal path to catch the target according to its foreseeing movements as shown in Figure 4(b). The target changes direction to northwest once it senses that the chaser is moving toward itself as shown in Figure 4(c). The target changes direction to north again as the chaser is getting closer. The chaser changes the direction dynamically to north at once where the target may head accordingly as shown in Figure 4(d). Eventually, the target is caught by the chaser with the 3D real time chasing algorithm as shown in Figures 4(e) (front view) and 4(f) (top view).



(a) Initiation (b) Target moves toward north.

(c)Target changes direction to northwest. (d) Target changes direction to north.

(e) Target is caught by chaser. (Front view) (f) Target is caught by chaser. (Top view)

Figure 4. Illustration of chasing system in a 50×50×50 volume.

4. Conclusion

In this paper, Jan's shortest path algorithm on 2D planar cells is extended to the volumes. The main achievements of these algorithms are to implement the 3D shortest path searching for multiple pairs based on the concept of the aircraft domain to avoid collision and real time chasing in the volumes.

We believe that this work will inspire further studies on the shortest path searching and chasing in quadric surface. Furthermore, our proposed algorithm is implemented based on the assumption that the aircraft's speed is constant, but it is impractical and the speed-varying system should be addressed in the future.

References

[1] C.Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. on Electron. Computer*, vol. EC-10, pp. 346-365, Sep. 1961.

[2] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. on Computer*, vol. C-23, pp. 907-914, 1974.

[3] J.H. Hoel, "Some variations of Lee's algorithm," *IEEE Trans. on Computer*, vol. C-25, no. 1, pp. 19-24, Jan. 1976.

[4] Gene Eu Jan, and Ki-Yin Chang, "An Improved Lee's Algorithm on Electronic Maps," *2002 International Computer Symposium*, National Dong Hwa Univ., Hualien, Taiwan, pp. 776~786, Dec. 2002.

[5] T.Y. Li, L.K. Gan, and C.F. Su, "Generating customizable guided tours for networked virtual environments," *In Proceedings of 1997 National Computer Symposium*, Taiwan, 1997.

[6] T.Y. Li, and P.Z. Huang, "Motion planning for a humanoid walking in a 3D space," *In Proceedings of 2001 National Computer Symposium*, Taiwan, 2001.

[7] G.E. Jan, K.Y. Chang, and J.S. Wu, "The planning of a 3D shortest path in a volume," *Trans. on AASRC*, vol. 35, no. 2, pp.197-202, 2003.

[8] J.H. Beattie, "The future of electronic chart in merchant ships," *The Journal of Navigation*, vol. 48, no. 3, pp. 335-348, 1995.

[9] J. Dawson, "Digital charting, now and in the future," *The Journal of Navigation*, vol. 52, no. 2, pp. 251-255, 1997.

[10] Gene Eu Jan, Ming-Bo Lin and Yung-Yuan Chen, "Computerized Shortest Path Searching for Vessels," *Journal of Marine Science and Technology*, Vol. 5, No. 1, pp. 95-99, June 1997.

[11] B.A. Colley, R.G. Curtis, and C.T. Stockel, "A marine traffic flow and collision avoidance computer simulation," *The Journal of Navigation*, vol. 37, no. 2, pp.232-250, 1984.

[12] P.V. Davis, M.J. Dove, and C.T. Stockel, "A computer simulation of multi-ship encounters," *The Journal of Navigation*, vol. 35, no. 2, pp. 347-352, 1982.

[13] F.P. Coenen, G.P. Smeaton, and A.G. Bole, "Knowledge-based collision avoidance," *The Journal of Navigation*, vol. 42, no. 1, pp.107-116, 1989.

[14] K.Y. Chang, G.E. Jan, and Ian Parberry, "A method for searching optimal routes with collision avoidance on raster charts," *The Journal of Navigation*, vol. 56, no. 3, pp.371-384, 2003.

[15] A. Stentz, "The focused D* algorithm for real-time replanning," *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, vol. 2, no. 2, pp.1652-1659, 1995.

[16] R.F. Dell, J.N. Eagle, G. Martins, and A. Santos, "Using multiple searchers in constrained-path, moving-target search problems," *Naval Research Logistics*, vol. 43, no. 4, pp. 463-480, June 1996.

[17] K.Y. Chang, G.E. Jan, and Ian Parberry, "Optimal search and interception system for multi-target on raster electronic charts," submitted to *Journal of Navigation*.