An Efficient Navigation Algorithm of Large Scale Distributed VRML/X3D Environments

Jinyuan Jia[1,2], Guanghua Lu[2], and Yuan Pan[2]

[1] School of Software Engineering, Tongji University,
Jiading District, Shanghai, P. R. China
csjyjia@yahoo.com.cn
[2] Zhuhai College of Jilin University,
Jinwan District, Zhuhai, P. R. China, 519041
{Lunikko, panyuanpy}@163.com

Abstract. Typical shortest-path search algorithm, e.g. Dijkstra algorithm, is difficult to implement in VRML/X3D world directly due to the simplicity of VRML/X3D programming. By using JavaScript with good cross-platform and compatibility with VRML, this paper proposed an efficient back-traceable climbing (BTC) based navigation algorithm, by improving Hill Climbing search algorithm with the destination oriented guidance and loop removal, and amplifying it with simple data structure and flexible interfaces. The BTC based navigation algorithm performs greatly better than Dijkstra algorithm in terms of efficiency, consumed memory and the number of accessed nodes. It also possesses the merits of simplicity, easy implementation and reliability. Experimental results also show that it can provide real-time virtual navigation services with enough precision for large scale VRML/X3D environment.

1 Introduction

Distributed virtual environment (DVE) becomes worldwide popularly with its immersion, interaction and imagination. VRML/X3D, as the second generation Web language after HTML, thoroughly changes the tedious interfaces and weak interactions of traditional Web applications. People can easily construct their own virtual world with VRML/X3D on Internet. However, with the size of virtual environments increasing, users usually feel lost during walkthrough in VRML/X3D environments and could not reach the destination they intended in advance. Too long time meaningless wandering with disorientation also makes users frustrated and loss the patience and interests to roam in virtual environments. Therefore, it is quite meaningful to provide efficient navigation algorithms for guiding users to walkthrough in large scale VRML /X3D environments. The research about this topic also attracts more and more attentions [1-6]. In this paper, we introduced an efficient navigation algorithm for guiding users to explore VRML/X3D environment effectively.

In Section 2, we summarize some related works concerning navigation algorithms of VEs. The proposed back-traceable climbing method is outlined roughly in Section 3 and illustrated in detail in Section 4 respectively. Experimental results and performance

R. Shumaker (Ed.): Virtual Reality, HCII 2007, LNCS 4563, pp. 244–252, 2007.

analysis are given in Section 5. The paper concludes in Section 6 by summarizing our contribution and discussing some thoughts concerning future work.

2 Related Works

Basically, navigation denotes the process of "wayfinding" to define an optimal path from one location to another in a VR environment which may be assisted by the underlying system. This is such a wide research area that it is difficult to summarize all of published works in this paper. Only typical or newest navigation techniques are addressed here.

A good survey on navigation algorithms of VEs was given by Ropinski *et al* [7]. Salomo *et al* [8] plans path for virtual avatars automatically and interactively with Probabilistic Road Map (PRM). Christie *et al* [8] specifies camera path according to some height information embedded into building models in VEs. Currently, there are three typical methods to search for the shortest path in VEs. The first one is classical Dijkstra algorithm [6][9-11]; the second one [12] is A^* based search method and the last [4] is grid based search method.

However, these three shortest path search algorithms are addressed to general virtual environments, not VRML/X3D environments, and they become impractical to implement due to the weak and simple programming capability of VRML/X3D provided. The navigation algorithm for VRML/X3D environments requires high efficiency, cross-platform, simplicity and easy implementation. Li *et al* [11] implemented their improved Dijkstra algorithm with Java Applet, its drawback is that the users with Window platforms have to download and install JVM inconveniently before running this algorithm. That might make those novice users lose their patience to go on walkthrough your VRML environments.

In this paper, we solved this problem by (*a*) improving traditional A^* algorithm with Back-Traceable Climbing maneuver; (*b*) making it simple enough for JavaScript programming; (c) implementing our proposed BTC based algorithm with JavaScript, Java Script possesses cross-platform and compatibility.

3 Major Idea of BTC Based Navigation Algorithm

3.1 Modeling Road Network Topologically

In realty, roadway not only has physical connectivity (without isolated points), but also logical connectivity, for two arbitrary points of roadway, you can always reach from one point to the other. Considering the fact that changing direction is required only at the crossing points of roadway, physical roadway is represented topologically with the road network model as shown in Fig. 1.

In the model above, the color circle denotes node, the number within the circle denotes the ID to identify this node, the connected line segment between two nodes is the edge and the number along the edge is the weight of path. In each node, two kinds of data are required to store, one is its absolute coordinate in VRML/X3D world, the other the lengths of all its incident edges. Generally, the nodes and their adjacency

relationships in road network are represented with the data structure, *adjacent list*, which could be provided only in Java, C++ and other advanced programming languages.

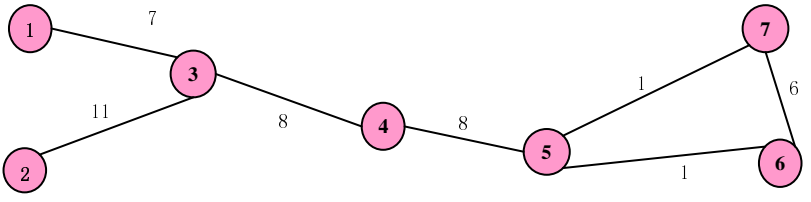

Fig. 1. Road network Model for Roadway

3.2 Heuristic Searching with Hill Climbing

Heuristic search is a search strategy to consider the applicable knowledge of problem space, dynamically determine the priorities of rules and choose the best rule of problem solving. Usually, it need to establish the *Open* list to record those nodes which have been produced but not traversed, and the *Closed* list to record those un-traversed nodes for maintaining the node data used in searching procedure.

Hill Climbing is a classical heuristic search strategy, at next step of search, the best node always is chosen to expand. It does not preserve the subsequent nodes and father of current node, during the path-searching, it only searches partially the solution state space and reduces the cost of dynamical maintenances by canceling the *Open* list. Thus, it possesses the merits of low time complexity, high efficiency and low memory cost. However, due to no historical records preserved, it has no back-tracing and restoring mechanism in case of "apexes", "basins", "ridges" and so on. So it easily falls into an optimal solution locally and even is unable to reach the final solution globally. Only back-traceable climbing can help overcome the drawback of traditional heuristic search with blind climbing.

3.3 Evaluation Function

In our proposed algorithm, the heuristic searching in AI is applied to navigation problem of road network. To an heuristic searching scheme [5], the best nodes for the optimal navigation path, it is important to design a good evaluation function, $f(m) = g(m) + h(m)$, for choosing the best node to expand at each step. Here, we improved two components $g(m)$ and $h(m)$ respectively. Firstly, $g(m)$ is set as the weight of N_mN_{m-1}, the edge from the node N_m to its father N_{m-1}, instead of the total path from the starting node to the current node N_m. Secondly, the distance $Sqrt((x_1 - x_2)^2 + (x_1 - x_2)^2)$ from the current node to destination is chosen as $h(n)$, and is further simplified as, $h(n) = |x_1 - x_2| + |y_1 - y_2|$ for removing computation of square roots. The function h(m) is to guide the searching to the nodes more hopeful to reach destination and avoid its blind searching along locally optimal directions. The function g(m) is to prevent the searching repetitively along wrong paths or loops.

3.4 Back-Traceable Climbing Maneuver

The back-traceable climbing maneuver is employed in our algorithm to avoid tramping into a locally optimal solution. our algorithm design a new data structure, valid node list VNL, instead of the open list and close list of traditional heuristic searching methods, to save those traversed nodes and current nodes which are chosen to expand at next step. Comparing to open list and closed list in traditional heuristic searching, VNL does not need recording all the nodes of road network and sorting priority of the list of expandable nodes, and thereby, dynamical maintenance operations are reduced a lot. Furthermore, two definitions are given at first to illustrate our BTC based navigation algorithm.

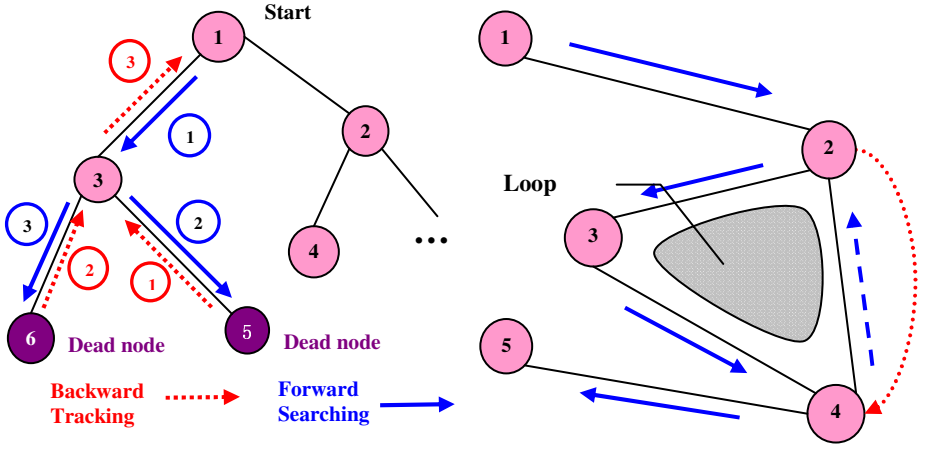

Fig. 2. Back-traceable Climbing Algorithm and Loop Removal

Definition 1. Dead Node is the node which has either no subsequent nodes or whose subsequent nodes all have been included in VNL already. If such dead nodes be found during path searching, then back-track processing has to do.

Definition 2. Active Node is the node which is not dead nodes and does not appear in VNL as well. During path searching, all active nodes can be chosen as candidates to expand at next step.

Two key points of BTC based navigation algorithm are how to give the back-tracking and ending condition of heuristic searching for the navigation path.

Here, the back-traceable condition of heuristic search is given as follows:

If the subsequent ones of current node all are dead nodes, then, go back (back-tracking) to its father node and expand. The detailed back-tracking process is depicted as in Fig. 2. The ending condition is given as follows:

- If the current node just is the destination, then searching succeed and return;
- If VNL becomes empty, then the searching fails and exits.

3.5 Loop Removal

Since the BTC based method only can give a locally optimal solution (not globally optimal solution) at each step, it cannot avoid the loop during searching for path. Therefore, it is necessary to judge where loops form and remove them from VNL in time during BTC based path searching, here, a simple loop removal method is proposed as follow. For the current node N_c and a subsequent node of it N_s, let us check if a loop can be formed after N_s is added and how the loop can be removed from current path in advance. A simple loop determination and loop removal method is suggested as the following steps:

(1) *search* N_s in VNL = $\{N_i\}$, $i = 1..m$, by comparing it with each element N_i;
(2) *if* it is not found in VNL, *then return* with 'no loop found', *else* do following loop removal processing:
   a) *locate* its position in VNL, say $N_s == N_k$ ($k \leq$ m);
   b) *delete* all the nodes (N_{k+1}, N_{k+2},…, N_m) behind N_k from VNL;
   c) *append* N_c into VNL;
(3) *fetch* next active subsequent node N_{ss} of N_c, *goto* (1).

4 Algorithm Description

4.1 Data Structures Based on VRML

Based on the script of VRML, two data structures are designed specially for road network as shown in Fig. 5, one is *node list* to store geometric information of all the nodes, the other is *adjacent edge list* to represent the adjacency relationships of two connected nodes (edges).

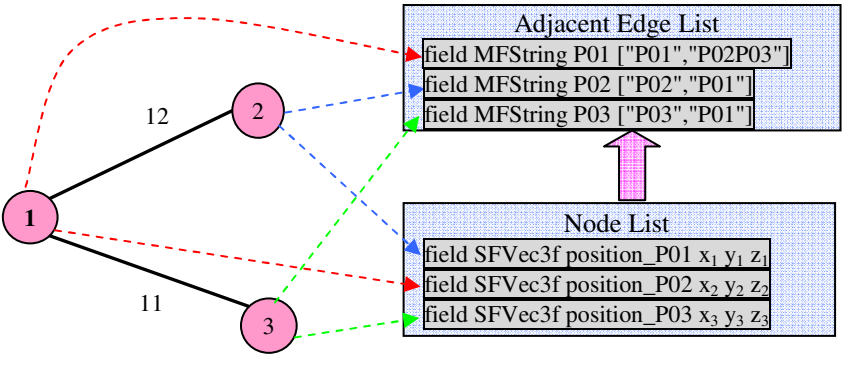

Fig. 3. Two data structures: (1) Adjacent Edge List (AEL) and (2) Node List (NL)

1) MFString, a field variable to store multiple string variables, can be used to store the adjacency relationship of any two connected nodes. Here, the array MFString[0..*n*] is used to represent the adjacent edge list, MFString[0] notes the VRML node P01 itself. MFString[1], field MFString P01

["P01","P02P03"], notes the two adjacent nodes P02 and P03 (two edges) of P01. MFString[2], field MFString P02 ["P02","P01"], notes the only one adjacent node P01 of P02. MFString[3], field MFString P03 ["P03","P01"], notes the only adjacent node P01 of P03.

2) SFVec3f, a VRML variable to store the coordinates of a 3D point (*x, y, z*), can be used to represent the location coordinates of a node for road network, for example, field SFVec3f position_P01 x1 y1 z1 notes the 3D coordinates (x1, y1, z1) of P01.

4.2 Computing Self-intersections of a Single Canal Surface

Based on BTC maneuver and VRML data structures, the algorithm of near shortest path search is sketched roughly as follows.

Notes: the beginning node: *start*, the ending node: *destination*, the current node: *c*, the subsequent node of *c* is *i*.

Step1【Initialization】VNL = {*start*} ; *c* = *start*;

Step2【Judge if VNL is empty】**if** VNL == Ø **exit**; 【Searching fails】

Step3【Judge if searching succeed】**if** *c* has any active subsequent node *i*, **get** one of them;

if *i* == *destination*, **return**; 【searching succeed】

Step4【Judge if active nodes exist】**if** *c* has no active subsequent node,

then【backtracing】

mark *c* as the dead node;

delete *c* from VNL;

goto Step 2;

Step5【Judge if loop met】**if** *i* can be found in VNL,

then【loop removal】

delete all the nodes behind *i* in VNL;

append *c* into VNL;

get the next active subsequent node *i* of *c*;

Step6【go forward】**if** *destination* can be found from the field of adjacent node of *i*,

then append *i* into VNL; *c* = *i*; **goto** Step 3;

Step7【expand the best subsequent node of *c*】

append the active subsequent node *i* of *c* with the smallest f(*i*) into VNL;

c = *i*; **goto** Step 3■

5 Experimental Results and Performance Analysis

An example, a transportation map of Zhuhai City, Guangdong, P. R. China as shown in Fig. 4, was chosen to implement with Java Script. and a road network with 153 nodes totally was created artificially, in which, the red circles correspond to the nodes of Zhuhai road network. To evaluate the performance of our proposed virtual

(1) (2)

Fig. 4. (1) Road network of Zhuhai City, (2) Virtual Zhuhai Campus of Jilin University

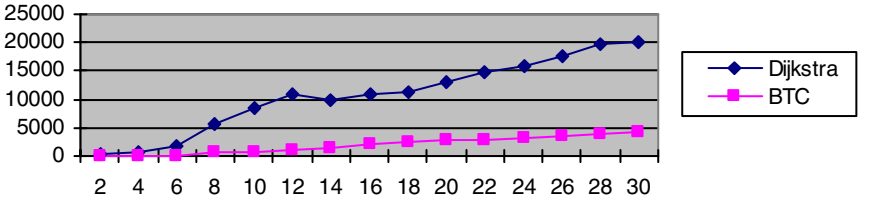

Fig. 5. Comparison on the Number of Accessed Nodes

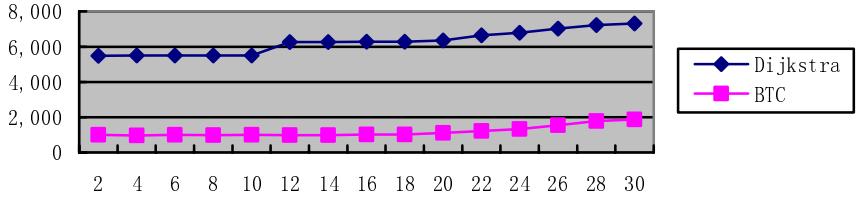

Fig. 6. Comparison on Memory Costs of Two Algorithms

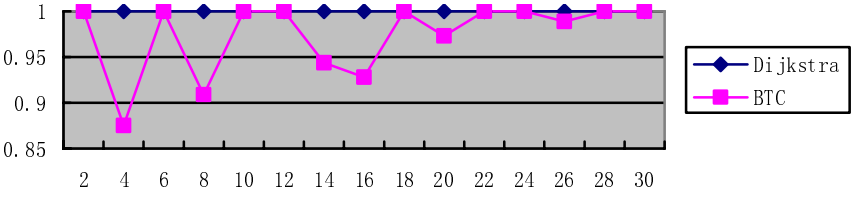

Fig. 7. Comparison on the Exactness of Resulting Paths

navigation algorithm, both classical Dijkstra algorithm and our own algorithm have been implemented with Java script programming at Pentium 4, 2.8 GHz CPU and 1GB RAM. The comparing experimental results and performance analysis are illustrated from Fig. 5 to Fig. 7 in terms of the number of accessed nodes, consumed memory and exactness respectively.

The following notations are given for performance analysis:

BL = Total length of the path generated by BTC algorithm;
DL = Total length of the path generated by Dijkstra algorithm;
Exactness = BL / DL * 100%.

Above experimental data as shown in Fig. 5 imply that BTC has strong directionality and only deal with the nodes and their sub-nodes along the route oriented to the destination during searching. Thereby, the total number of nodes accessed by BTC should be proportional to the length (the number of nodes) of the final path. However, the number of nodes traversed by Dijkstra algorithm should increase exponentially with the size of road network, and the number of nodes traversed by BTC algorithm is extremely less than by Dijkstra algorithm as shown in Fig. 6. Similarly, the memory cost of BTC is far from the memory cost of Dijkstra algorithm as well, as shown in Fig. 7, it always maintains a low level stably without leaping acutely with the size of road network increasing, for example, the memory cost of Dijkstra algorithm for processing the path with more than 10 nodes is 1000KB (about 1MB) more than for processing the path with less than 10 nodes. But, BTC removes dynamical maintenance of Open list by improving traditional climbing manner, and thereby achieves much lower memory cost than Dijkstra algorithm, however, the next node at each step expanded by BTC algorithm only is a locally optimal solution currently, that might not become a globally optimal solution finally. Usually, the exactness of BTC could not reach 100%, as shown in Fig. 8, its exactness fluctuates seriously at the topology of road network for a small scale virtual environment, however, the fluctuation becomes slightly and the exactness goes increasing stably with VRML environment being larger and larger.

Classical Dijkstra algorithm is a brute force searching algorithm, it expands the next node blindly according to equal probability criterion for all subsequent nodes, without considering the orientation and position of destination its time complexity is

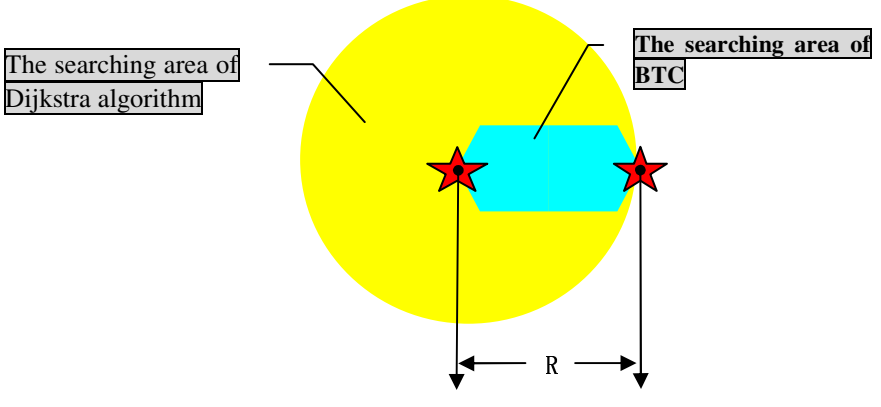

Fig. 8. Searching areas of Dijkstra algorithm and BTC

$O(n^2)$, where n is total number of nodes of road network, so the search area of Dijkstra algorithm is the yellow circle with the beginning node as its center and the distance to destination as its radius in Fig. 8.

6 Conclusion

We proposed a high efficient BTC based navigation algorithm and implemented it in our big VRML world (Virtual Zhuhai Campus of Jilin University, http://soft.jluzh.com/yanshi/school/index.htm) by using JavaScript purely. Thereby, it owns the merits of JavaScript programming, cross-platform and free-plugin, and facilitates users to login VRML world and walkthrough with navigation services on Internet conveniently. Due to the limit of JavaScript itself, this algorithm may not achieve the most ideal effects. However, it can get the optimal or near-optimal paths in most cases. Huge practical tests also have shown that this algorithm can completely satisfy both requirements of real-time walkthrough and navigation exactness on Internet.

One important advantages of this algorithm is the low cost and fewer traversing nodes, but it just reduces the exactness of the resulted navigation path in the case of loops and backtracking. Theoretically, this drawback could be overcome by devising a better evaluation determinant. It will be regarded as our future work.

References

1. Knight, C., Nunro, M.: Virtual but Visible Software. In: Proceedings of IV00, International Conference on Information Visualization. IEEE CS Press, Washington, DC, USA (2000)
2. Chittaro, L., Coppola, P.: Animated Products as a Navigation Aid for E-commerce. In: CHI00. Proceedings of Computer Human Interaction, pp. 107–108. ACM Press, Netherlands (2000)
3. Bowman, D.A., Koller, D., Hodges, L.F.: 3D User Interface Design. In: Course Notes of SIGGRAPH '00, ACM Press, New York (2000)
4. Chittaro, L., Ranon, R., Ieronutti, L.: Guiding Visitors of Web3D Worlds through Automatically Generated Tours. In: Proceedings of Web3D, pp. 27–38. ACM Press, New York (2003)
5. Bowman, D.A., Koller, D., Hodgesm, L.F.: 3D User Interface Design. In: Course Notes of SIGGRAPH '00, ACM Press, New York (2000)
6. Benjamin, Z.F., Charles, N.E.: Shortest Path Algorithms: An Evaluation Using Real Road Networks, Transportation Science, 65–73 (1998)
7. Ropinski, T., Steinicke, F., Hinrichs, K.: A Constrained Road-Based VR Navigation Technique for Travelling in 3D City Models
8. Christie, M., Languenou, E., Granvilliers, L.: Modeling camera control with constrained Hyper tubes. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, Ithaca, NY, pp. 618–632 (2002)
9. Dijkstra, E.W.: A note on two problems in connection with graphs. Numerical Mathematics, pp. 269–271 (1959)
10. Jang, S.D.: The Use and Audience Research of Interactive Multimedia Tours Guide Systems in Museums. Masters Thesis, Institute of Communication Technology, National Chiao Tung University, Hsinchu, Taiwan, R.O.C. (1994)
11. Li, T.Y., Gan, L.K., Su, C.F.: Generating Customizable Guided Tours for Networked Virtual Environments. In: Proceedings of NCS'97 (1997)
12. Luger, G.F.: Artificial Intelligence Structures and Strategies for Complex Problem Solving, Fourth edn. China Machine Press (2004)