Appear in Proceedings of the IEEE International Conference on
Robotics and Automation (ICRA), 2001

Path Planning with Incremental Roadmap Update for Large Environments

Tsai-Yen Li
Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
li@cs.nccu.edu.tw

Chih-Ching Chang
Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
s8522@cs.nccu.edu.tw

Abstract

Recent research results suggest that one can incorporate motion-planning techniques into the control loop of 3D navigation or tele-operation for more efficient navigation. However, the motion planner with this approach may not scale up well for large workspaces. In this paper, we propose a novel approach to overcome this scalability problem. We limit the region of interest for path-finding to a window around the current robot configuration and incrementally update the roadmap in this window as the robot moves. In order to make the roadmap update efficient enough for interactive applications, we adopt a recently proposed data structure, called Rapidly-Exploring Random Tree (RRT), to reduce the run-time cost of building the connectivity roadmap. The incremental path planner has been implemented in Java and incorporated into a Java3D-based VRML browser. We compare the performance of this improved planner with the previous one for workspaces of various sizes and analyze the bottlenecks of maintaining such a roadmap. By extending the planning techniques to large workspaces, we believe that this type of intelligent navigation or tele-operation control will inspire better user-interface design and further researches in planning for large or unbounded worlds.

1. Introduction

The rapid development of computer technologies and the Internet have revolutionized many traditional applications. For example, many virtual reality (VR) and robotics applications are moving toward distributed computing model on the Internet. The concepts of tele-presence, Internet robot, and virtual laboratories become active research topics in the past few years. However, we believe that there still exist problems that need to be solved before these applications can really take off. For example, for novice users equipped with a 2D mouse, it remains a great challenge to precisely perform remote control or walkthrough navigation in a large or complex environment.

No matter that a user is controlling a robot remotely or a viewpoint locally, some collision detection routines need to be applied to ensure safe motions or to improve realism. However, under such a control mode, a user (even an expert user) often runs into a situation where the controlled object or viewpoint gets stuck at certain locations of the scene. Several maneuvers are often required to get them out of this kind of situation. Users often feel frustrated with this level of control, especially when the control rate or frame rates are low for large environments. In [11], we proposed an intelligent user interface for navigation control or teleoperation that incorporates motion-planning techniques to assist users in performing low-level control. It voluntarily generates collision-free paths to avoid collisions with obstacles in the environment while the user is directing the motion. Experiments showed that it is an effective interface that can reduce the average execution time for accomplishing a given navigation task by seventy-six percents.

However, a main limitation of this planner is that it does not scale up well with as the size of the workspace and the number of obstacles increase. In this planner, one builds a randomized roadmap to capture free-space connectivity by sampling enough configurations in a preprocessing step. However, building and storing the roadmap become infeasible when the size of the workspace becomes very large. In this paper, we extend this intelligent navigation/control interface to consider the case of large workspaces. We propose to incrementally update the roadmap in a dynamic manner by maintaining only the nodes in a moving window around the robot during its navigation. Maintaining the roadmap in such a window is a challenging task because the computation time needs to be short enough for incorporating planning into the control loop of an interactive system. We will describe that data structures and algorithms that allow us to efficiently update the dynamic roadmap at run time.

The rest of the paper is organized as follows. We will review related work in motion planning and intelligent user interface in the next section. We will then review how path planning is incorporated into the user control loop to assist a user in performing navigation and address the issues of bringing the problem on-line for large

workspace in Section 3. We will then show how we approach this on-line planning problem with the Rapidly-Exploring Random Tree (RRT) structure and how the roadmap is updated as the robot moves in Section 4. We will then show the details of our implementation in Section 5, and the experimental settings, results, and analysis in Section 6. Finally, we will conclude our work in the last section.

2. Related Work

The researches pertaining to our work fall into two categories: *3D intelligent user interface* in Computer Graphics and *path planning* in Robotics. More specifically, our work is on applying path planning techniques to the design of teleoperation or navigation interfaces. Therefore, we review related work in these two areas in this section as follows.

2.1. 3D intelligent user interface

Most of the developments in 3D user interface design use the direct manipulation metaphor that is shown to be more comprehensible, predictable, and controllable than the delegation types of intelligent user interfaces in several application domains. However, it is still under debates which metaphor is more effective in general.[14] In fact, effectiveness would greatly depend on the application types, user sophistication, and task complexity. For example, some people may prefer to sit back and take a guided tour when visiting a new environment while other adventurous people may prefer to have a full control of navigation.

Many intelligent user interfaces have been proposed in the past, but most of them are not for 3D manipulation applications.[12] Exceptions include using motion-planning techniques to provide task-level controls. For example, Drucker and Zeltzer [4] argue that a task-level viewpoint control is crucial for exploring virtual scenes such as virtual museums because the users should concentrate on scene viewing instead of be distracted by low-level navigation controls. Li, et al. [10] also proposed an auto-navigation system that generates customized guided tour based on high-level user inputs. Kuffner [7] also utilizes fast path-planning techniques for real-time animations. These researches use geometric reasoning techniques as a tool for control delegation from a third-person view, which is different from our first-person view in scene navigation. Other works also suggest using vector fields such as force fields to constrain 3D navigation with a first-person view [5][15]. However, they are scene-specific design tools that do not incorporate planning techniques.

2.2. Randomized path planning

In this paper, we define the navigation problem as an example of the path-planning problem (or the so-called *Piano Mover's Problem*) that has been well studied in

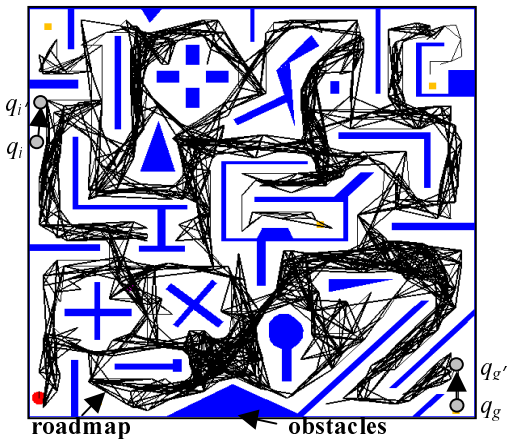

Figure 1: a sample probabilistic roadmap

Robotics in the past three decades[9]. Under the curse of dimensionality, the computational complexity of the problem is exponential in the degrees of freedom (DOF) that the moving object possesses.[13] Most efficient complete path planners exist only for three or four dimensional configuration space (C-space). The methods bases on artificial potential fields are well-known examples capable of solving 2D path-planning problems in fractions of a second.[2] For problems with high dimensional C-space, an effective planning scheme called *random sampling scheme for path planning* has been reported to be effective in solving practical problems in various applications[1]. The *probabilistic roadmap planner*[6], a special version of the randomized planners, has been adopted in our previous work to assist user navigation task. Many forms of roadmaps that try to capture the connectivity of freespace effectively have also been proposed. [8]

3. Intelligent Control Interface

3.1. New interface with path planning

A new intelligent control interface has been proposed in our previous work to incorporate efficient path planning into the control loop of the navigation. [11] The new interface is effective for two reasons. First, the average planning time in each control loop is rather short (tens of ms). Consequently, the frame rate is still high enough for smooth and responsive navigation. Second, the overall execution time for accomplishing a given task (e.g. visiting a sequence of locations in the workspace) is greatly reduced because many unnecessary maneuvers have been eliminated.

In order to generate a useful path in a short amount of time, we adopt the path-planning algorithm with the probabilistic roadmap approach proposed in [6]. In its preprocessing step of this approach, the algorithm builds a roadmap that captures the connectivity of the freespace such as the one shown in Figure 1. User mouse inputs are

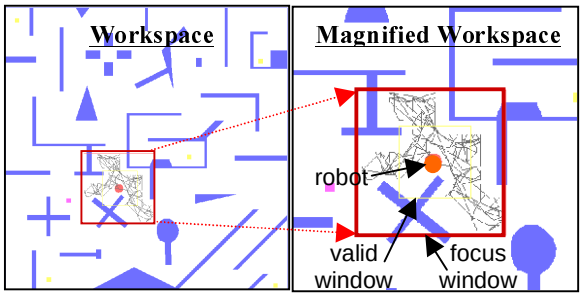

Figure2: RRT, valid window, and focus window

transformed at run time into desired goal configurations. The path planner is evoked for computing a collision-free path from the current configuration to the goal configuration whenever there exist no straight-line path between them. This step involves searching the connectivity graph of the roadmap to connect these two configurations. An optional post-processing step smoothes the found path into a shorter one before it is sent for execution. In summary, the system uses motion-planning techniques to voluntarily assist a user in navigating through difficult areas of a virtual scene.

3.2. Extensions to large environments

The traditional roadmap planners may not scale up well to deal with large-scale workspaces. The roadmap planner in our previous experiments is efficient for several reasons. First, we assume that the world is a static bounded workspace with known obstacle configurations. Second, collision detection, the most time-consuming routine in the planner, is made easy by looking up a pre-computed C-space. Third, the size of the roadmap that captures freespace connectivity is moderate, and the time for searching the roadmap (an undirected graph) is insignificant.

The first factor is not directly affected by the size of the workspace while the second and third factors may make the planner impractical for interactive uses as the world becomes very large. For example, although the C-space obstacles are computed off-line in a preprocessing step, the space for storing the C-space obstacles may increase to an unacceptable extent as the size of the world increases. The size of the roadmap and the time for searching the roadmap also increases significantly as the size of world becomes large.

4. Incremental Roadmap Update

The key to extending the new interface to large workspaces lie on the facts that the path-planning problem for our navigation control usually can be confined in a local region around the current robot configuration. No matter how large the whole world is, two consecutive configurations are expected to be close to each other under regular navigation operations. Thus, we can define a window called *focus window* in the workspace around the robot

Generate_RRT(x_{init}, K)
1. T.init(x_{init})
2. **for** k=1 **to** K **do**
3. x_{randt} ← Random_Configuration();
4. x_{near} ← Nearest_Neighbor(x_{randt}, T);
5. x_{new} ← New_Configuration(x_{near}, x_{randt});
6. T.add (x_{near}, x_{new});
7. **Return** T

Figure 3: algorithm for generating RRT

for maintaining a roadmap as shown in Figure 2. The size of the window should be large enough to enclose the current and next configurations as well as the path connecting them. We also define another smaller window called *valid window* inside the focus window. Whenever the robot exits the valid window we update both windows according to the new configuration. The size of the focus window determines the size of the roadmap while the size of the valid window determines how often the dynamic roadmap is updated.

The problem now becomes how to maintain the roadmap in the focus window as the robot moves. Traditional roadmap planners build the roadmap for the entire C-space in a one-time preprocessing step. The typical time for building such a roadmap is several seconds for a moderate world, which is too long to be used in a control loop. We need to have a more efficient method to represent and update the roadmap in an on-line manner. In this section, we first describe a special form of roadmap called RRT that we have adopted in our on-line planner. We will then show how we extend this data structure to consider on-line deletions and additions of nodes in RRT. Finally, we will describe how we use a 2D range search tree to speed up collision detection routines for on-line collision queries.

4.1. The RRT data structure

Recently a new data structure called Randomized Rapidly-Exploring Random Tree (RRT) has been proposed as one type of roadmap for path planning.[8] It has been shown to be an effective method for evenly exploring the freespace. The algorithm for this method is sketched in Figure 3. The goal of the algorithm is to build a tree T of configurations that represent the freespace. Initially the tree consists of the initial configuration (x_{init}) only and new configurations are added incrementally into T. For each of the K configurations to be added into T, we first generate a random configuration (x_{rand}) as a target for growing the tree (line 3). Then we try to find the nearest configuration (x_{near}) in the tree from the target (line 4). From this nearest configuration we grow the tree toward the target as far as possible until the target is reached or an obstacle is encountered (line 5). The farthest configuration (x_{new}) that can be reached is then added into the tree (line 6).

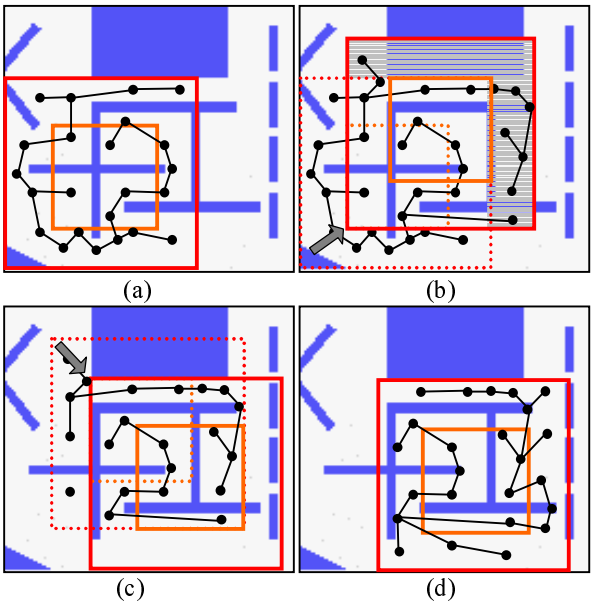

Figure 4: RRT incremental update process as the focus window moves

We have chosen the RRT algorithm for two of its properties. First, the number of links amongst nodes is small since it uses a tree structure instead of a general interconnected graph as in the traditional roadmap approach. As a result, the storage requirement and computation time for updating a tree is also smaller. This property is crucial for the interactivity requirement in our application. Second, the generated tree typically is a good representation of the freespace. It has been shown that the RRT approach tends to explore area that hasn't been explored yet [8].

4.2. Updating RRT in the focus window

When the robot is at the initial configuration, the path planner builds an RRT of K nodes in the focus window such as the one shown in Figure 4(a). When the robot exits the valid window, we update the RRT according to the new configuration (Figure 4(b)). The update considers the differences between the old and the new regions that include deletion and addition of nodes from the tree. For the example in Figure 4(b), as the robot move northeast, we need to delete the nodes in the L-shape region D at the lower left corner and add new nodes into the flipped L-shape region A at the upper right corner.

In order to support dynamic update of the RRT, we modify the original RRT structure to support the forest data structure and its associated operations. The original RRT is a tree structure rooted at the initial configuration. With our incremental update, this is no longer true since deleting nodes may cause a tree to split into several subtrees and adding nodes may merge two sub-trees into a single tree. For example, in the RRT in Figure 4(c), the original tree is split into three subtrees. As the robot moves to another configuration that causes the update in Figure 4(d), the RRT merges two subtrees into one. A given number of random configurations are generated and added into the new region one by one. When a new node is added, we use the Generate_RRT algorithm in Figure 3 to connect the new node to each tree in the current forest. Two trees are merged into one if they can be connected via the new node. In the merge operation, we also need to reverse the parent-children relations for the nodes along the path from the merging node to the root of the tree.

4.3. Querying obstacles in the focus window

Collision detection is considered to be the most fundamental and time-consuming routine in path planning. In our current implementation, we assume that the robot can be represented as an enclosing circle of certain radius so that we can simplify each collision check to a table lookup in a 2D bitmap representing the configuration space (1:obstacle, 0:freespace). Given a geometric description of obstacles in a workspace, we can build the bitmap by a common scan-line algorithm in 2D computer graphics. However, when the workspace is large, it becomes impractical to maintain such a bitmap for the whole configuration space. Consequently, we must also update the bitmap in the focus window on the fly as it moves. In order to update the bitmap, we have to know which obstacles overlap with the new region in the focus window. This is a standard 2D range-query problem in computational geometry.[3] We use a standard 2D range tree to organize the bounding boxes of the obstacles so that we can answer the question of which obstacles intersect with the L-shaped region as quickly as possible even for a large workspace.

5. Implementation

The path planner with static and dynamic roadmap updates has been fully implemented in Java. We incorporate the planner into a 3D navigation interface by modifying an open source VRML browser implemented based on the Java3D SDK library. This SDK and the VRML browser are all available for FTP on the public domain.[16] We enhance the browser with our implementation of collision detection routines. These collision checks are evoked for configuration update to prevent potential collisions even when the path planner is not used.

The workspace for the virtual environment is modeled as a discrete 2D space of certain resolution (e.g. 128x128). To support dynamic roadmap update, we use a window size of 16x16 and 32x32 for the valid window and focus window, respectively. The total number of nodes generated for the RRT in the focus window is about 250. The number of nodes successfully added into the RRT depends on the number of nodes deleted during an update so that the total number of nodes in the focus window remains the same.

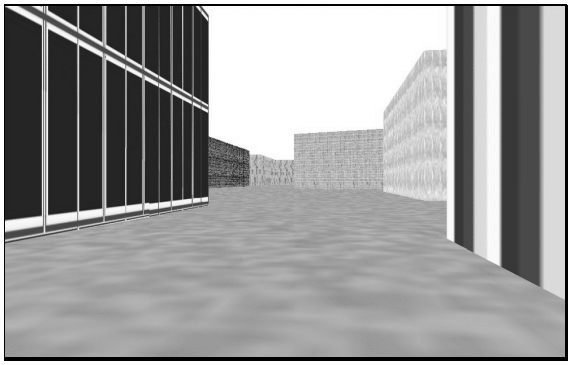

Figure 5: a snapshot of the VRML browser during the navigation experiment

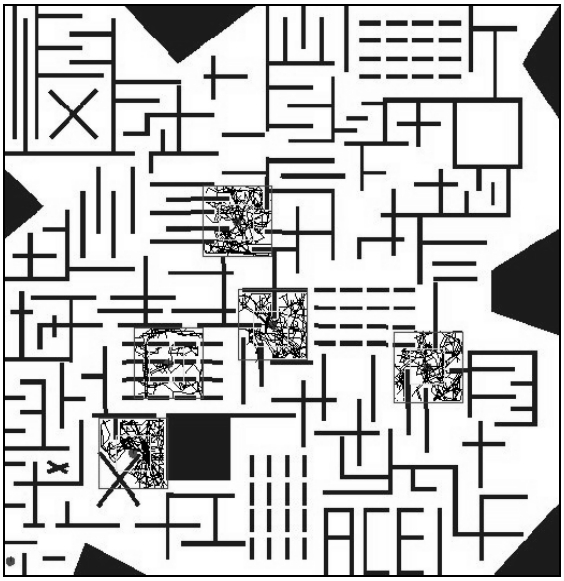

Figure 6: examples of RRT's in the focus window at different times in a large workspace

6. Experiments

Our experiments aim to compare the performance of the new planner with dynamic roadmap update to that of the original static planner. Because the benefits of incorporating planning into the navigation interface have been demonstrated in our previous work, our experiments only focus on observing the impact of maintaining a dynamic roadmap on-line.

6.1. Experimental settings

The experiments ran on a regular PC with a Pentium III 550 processor. We first use a small world (a maze-like environment as shown in Figure 1) of size 128x128 cluttered with 50 obstacles to compare the static roadmap planner and the dynamic roadmap planner. Statistic data are collected for a task of navigating through a similar path of about the same size. The number of planning requests is about the same for the entire navigation experiment. We then test the dynamic roadmap planner in a large world of size 256x256, consisting of 300 obstacles (as shown in Figure 5 and Figure 6) to see how world size affects planning time.

Table 1: performance comparison for different planners in different environments

	P1 with small world	P2 with small world	P2 with large world
preprocessing time (ms)	4062	94	125
no. of total steps	1215	1202	3165
no. of planning requests	16	25	42
time for each path search request (ms)	275	50	83
time for each path smoothing (ms)	186	49	81
no. of window updates	N/A	74	89
time for each window update (ms)	N/A	41	22

6.2. Experimental results

We summarize the experimental results in Table 1. The second column lists the data for the static roadmap planner (P1) with the small (128x128) world while the third column is for the dynamic roadmap planner (P2) with the same world. The fourth column shows the data for P2 with a large (256x256) world.

We have the following observations from the data in Table 1. The preprocessing time for P1 is significantly larger than P2 since it builds a complete roadmap for the entire C-space. For a same workspace, the planning time (searching the roadmap for a path and smoothing the found path) for P1 is larger than that for P2 mainly because the size of the static roadmap is larger than the dynamic roadmap. However, P2 needs to spend extra time to maintain the dynamic roadmap as the focus window moves. For example, in the second case (P2 with the small world), the roadmap is updated 74 times during the experiment. The average time for each update is 41ms, which is insignificant compared to the time for planning and graphics rendering. Examples of RRT in the focus window at different locations are shown in Figure 6. Experiments conducted in the third case (for the large world) show that the average update time remains at the same degree as in the case of the small world. The computation for each update includes updating C-space bitmap, deleting nodes and adding nodes into RRT. Our experiments show that the most significant computation (about 90%) in each update is for node additions. About one fourth of nodes (60) are deleted and about the same number of nodes are added in each update.

6.3. Discussion

The main challenge of extending path planning to a large

world is on the efficiency of maintaining a dynamic roadmap around the robot. Ideally, we hope that by incrementally updating the roadmap, the complexity of the planner can be independent of the world size (and the number of obstacles in it). Unfortunately, such an ideal situation cannot be achieved. First, the number of nodes in the dynamic roadmap does not change much as the focus window moves. Therefore, the time complexity for path queries is constant and independent of the world size. Second, the number of nodes deleted and added into the dynamic roadmap is also independent of the world size. The only module that depends on world size is the collision detection routine. Assume that we do not have explicit representation of the entire C-space (since the world could be very large), and then we have to perform collision checks at run time, which depends on the number of obstacles in the world. In our current implementation, we use a 2D range tree to organize the bounding boxes of the obstacles. This data structure is used for faster overlap queries when we need to update the bitmap for collision checks as the window moves. However, the time complexity for the standard 2D range query algorithm is $\log^2 n$, which is no longer constant.[3] Fortunately, our experimental data show that the computation time for this operation is less than 5% of the overall update time (1.27ms and 1.56ms for the small and large worlds, respectively). Therefore, this factor actually should not affect the performance until the world size becomes really huge.

Although path planning is typically considered a time-consuming task, the planning time for our problem is usually only small fractions of a second. This efficiency is mainly due to the fact that we always confine the planning in the focus window. On the other hand, the planner may fail to find a feasible path simply because the path does not lie entirely inside the window. Therefore, there exists a tradeoff between completeness and efficiency. The choice for the sizes of the valid window and the focus window in our experiments are determined empirically for now. We think it is a subjective matter to determine their sizes, and they should actually vary for machines of different processing power.

7. Conclusions

In this paper, we have proposed improvements to our previous work on designing an intelligent navigation interface for tele-operation and architectural walkthrough applications. We extend the planner to consider the case of large workspaces by proposing a planner that maintains a special form of dynamic roadmap (RRT) at run time. We update the roadmap confined in the focus window on demand as the robot moves. Our experiments show that the planner has the same degree of efficiency as the previous planner. The benefits of using this type of intelligent user interface (mainly on overall execution time) can therefore be extended to large workspaces. We believe that the proposed planning scheme is not only interesting for our navigation problem, but also intriguing for on-line planning problems with a large or unbounded world in robotics.

8. Acknowledgement

This work was partially supported by National Science Council under contact NSC 89-2218-E-004-009.

9. References

[1] J. Barraquand, L. Kavraki, J.C. Latombe, T.Y. Li, and P. Raghavan, “A Random Sampling Scheme for Path Planning,” *Intl. J. of Robotics Research*, 16(6), pp.759-774, Dec. 1997.

[2] J. Barraquand and J. Latombe, “Robot Motion Planning: A Distributed Representation Approach,” *Intl J. of Robotics Research*, 10:628-649, 1991.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 1997.

[4] S. M. Drucker and D. Zeltzer, “Intelligent Camera Control in a Virtual Environment,” *Graphics Interface’94*, pp.190-199, 1994.

[5] L. Hong, S. Muraki, A. Kaufman, D. Bartz and T. He, “Virtual voyage: interactive navigation in the human colon,” *Proc. of Computer Graphics (SIGGRAPH 97)*, pp. 27-34, 1997.

[6] L. Kavraki, P.Svestka, J. Latombe, and M. Overmars, “Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces,” *IEEE Trans. on Robotics and Automation*, 12:566-580, 1996.

[7] J.J. Kuffner and J.C. Latombe. "Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans,". *Proc. of CA '99: IEEE Intl. Conf. on Computer Animation*, Geneva, Switzerland, May 26-29, 1999.

[8] J. Kuffner, and S. LaValle, “RRT-Connect: An Efficient Approach to Single-Query Path Planning,” *Proc. of 2000 IEEE Intl. Conf. on Robotics and Automation*, May 2000.

[9] J. Latombe, *Robot Motion Planning*, Kluwer, Boston, MA, 1991.

[10] T.-Y. Li, J.M. Lien, S.Y. Chiu, and T.H. Yu, “Automatically Generating Virtual Guided Tours,” in *Proc. of the Computer Animation '99 Conf.*, pp.99-106, May 1999.

[11] T.-Y. Li, and H.-K Ting ., “An Intelligent User Interface with Motion Planning for 3D Navigation,” *Proc. of the IEEE Virtual Reality 2000 Conf.*, March 2000.

[12] M. Maybury and W. Wahster (eds), *Readings in Intelligent User Interfaces*, Morgan Kaufmann, CA.

[13] J.H. Reif, “Complexity of the Mover's Problem and Generalizations,” *Proc. of the 20th IEEE Symp. on Foundations of Computer Science*, pp. 421-427, 1979.

[14] B. Shneiderman and P. Maes, “Direct Manipulation vs. Interface Agents,” *Interactions*, 4(6):42-61, Nov. 1997.

[15] D. Xiao, R. Hubbold, “Navigation Guided by Artificial Force Fields, “ *Proc. of the ACM CHI’98 Conf.*, pp.179-186, 1998.

[16] The Java3D and VRML working group, URL: http://www.vrml.org/WorkingGroups/vrml-java3d