An Intelligent 3D Navigation Interface for Large Virtual Environments

Tsai-Yen Li and Chih-Ching Chang
Computer Science Department, National Chengchi University
64, Sec.2, Chih-Nan Road, Taipei, Taiwan 11623, ROC
e-mail: {li, s8522}@cs.nccu.edu.tw

Abstract

Recent developments on graphics hardware and software have opened up new 3D interactive applications on personal computers. Most of these advances aim to provide better and faster graphics rendering. Recent researches suggest that motion-planning techniques can be successfully incorporated into the navigation control loop to improve the efficiency of 3D navigation in an architectural environment. However, a main limitation of this approach is on the scalability of the motion planner for large virtual environments. 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 viewpoint and incrementally update the roadmap in this window as the user viewpoint moves along. In order to facilitate the incremental update of the roadmap, we adopt a new 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 and incorporated into a Java3D-based VRML browser. We report experimental results of this improved user interface on a large-scale virtual environment. Computational bottlenecks of maintaining such an interface at run time are also analyzed. By extending the planning techniques to large-scale virtual scenes, we believe that this type of intelligent navigation will inspire better interactive 3D user-interface design in the future.

1. Introduction

Due to the rapid evolution of computer hardware and software, two trends can be observed for virtual reality (VR) applications: the hardware platform is moving toward low-cost desktop PC's and the software architecture is moving toward distributed computing models on the Internet. The price for the next generation of display cards with interactive 3D acceleration is becoming more affordable than ever. The efforts of standardizing 3D graphics format, such as the Virtual Reality Modeling Language (VRML) language[22], also contribute to populate 3D applications. One can envision exciting opportunities for new applications with interactive 3D graphics on PC's. However, we believe that there still exist problems that need to be solved before VR applications can really take off. For example, for novice users equipped with 2D mouse, it remains a great challenge to precisely control walkthrough navigation in a complex virtual environment.

Most VRML browsers support architectural walkthrough type of navigation control with some optional collision detection functions. These functions increase the degree of realism by detecting collisions between the viewpoint and the environment in order to prevent the viewpoint from penetrating the obstacles. However, under such a navigation mode, a user (even an expert user) often runs into a situation where the controlled 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 navigation control, especially when the given frame rates are low for large virtual worlds.

This type of navigation control uses the direct-manipulation metaphor. This metaphor has been shown to be effective one for user interface design since the system behavior is more predictable than the one with intelligent user interfaces based on agent technologies. [20] However, we think the premise for this claim is that the user interface is responsive and the control is not too tedious. This premise may not hold for 3D interactive graphics. In [16], we proposed an intelligent user interface for navigation control aiming to taking advantages of both metaphors. This approach incorporates motion-planning techniques to assist users in avoiding obstacles by voluntarily generating collision-free paths to avoid collisions with obstacles. Experiments showed that it is an effective interface that can reduce the average execution time for accomplishing a given navigation task by 76 percents.

However, this planner may not scale up well. A main limitation of this implementation is on the size of the virtual environment and the number of obstacles in it. In the randomized roadmap approach used in the planner, one needs to sample enough configurations in the preprocessing step in order to build a good roadmap that captures the connectivity of the freespace. However, building and stor-

ing the roadmap become infeasible when the size of the virtual environment becomes very large. In this paper, we extend this intelligent navigation interface to consider the case of large virtual worlds. We propose to dynamically update the roadmap by maintaining only the nodes in a moving window around the viewpoint during navigation. Maintaining the roadmap in such a window is a challenging task because the computation time needs to be short enough for incorporating the computation into interactive navigation. We will describe that data structure and algorithms that allow us to efficiently update the dynamic roadmap at run time.

We organize the rest of the paper as follows. We will review some related researches in motion planning and intelligent user interface design 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 virtual worlds 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 viewpoint 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 and discuss future extensions in the last section.

2. Related Work

The researches related to our work fall into two categories: *3D user interface design* in the field of computer graphics and *path planning* in the field of robotics. More precisely, our work is on applying the path planning techniques learned in robotics to the design of 3D navigation interface. Therefore, we review pertaining work in these two areas in this section as follows.

2.1. 3D user interface design

Many researches in user interface are undertaken to invent new efficient ways to communicate with a computer and on evaluating the effectiveness of these interfaces. As the computer hardware for 3D acceleration becomes faster and affordable, VR-types of user interfaces for 3D virtual environments has become popular. For example, high-resolution Head Mounted Display (HMD), 3D tracking devices, data gloves, force feedback joysticks, and other haptic devices are all good examples of devices under active studies and development. New metaphors such as eyeball in hand, and flying vehicle in hand have shown to be effective metaphor in the context of virtual space exploration.[4] However, it remains a great challenge to design an effective interface to manipulate a 3D virtual scene only with a regular 2D mouse on a desktop computer.

Most of the aforementioned developments 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.[20] Now people tend to agree that effectiveness would greatly depend on the types of applications, users, and tasks at hand. 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 navigation control.

Although many intelligent user interfaces have been proposed in the literature, most of them are not for 3D manipulation. [17][18] Exceptions include using motion-planning techniques to provide task-level controls. For example, Drucker and Zeltzer [5] argue that a task-level viewpoint control is crucial for exploring virtual scenes such as virtual museums since the users should be allowed to concentrate on scene viewing instead of be distracted by low-level navigation controls. Li, et al. [15] also proposed an auto-navigation system capable of generating customized guided tour based on high-level user inputs. Kuffner [11] also utilizes fast path-planning techniques to assist 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 [7][8][21]. However, it is a scene-specific design tool and does not incorporate any planning techniques in it.

2.2. 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 the past three decades[13]. Under the curse of dimensionality, the computational complexity of the problem is exponential in the degrees of freedom (DOF) that the moving object possesses.[19] Most efficient complete path planners exist only for three or four dimensional configuration space (C-space). The methods bases on artificial potential fields are good examples that are reported to be able to solve 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]. A special version of planner with this random sampling scheme, called the *probabilistic roadmap* method[10], 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. [12]

3. Intelligent Navigation Interface

3.1. New user interface for path planning

In our previous work[16], we proposed a new interface

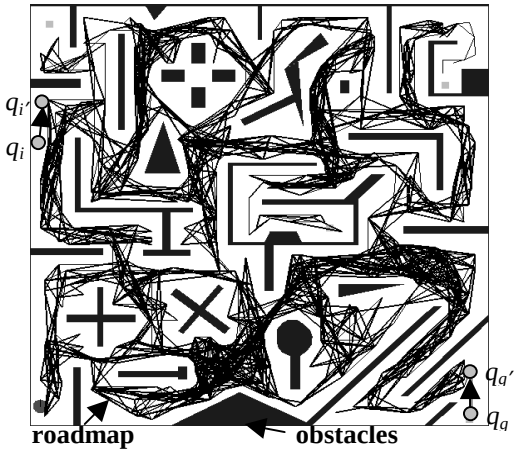

Figure 1. A sample probabilistic roadmap

scheme that incorporates an efficient path-planner into the control loop of navigation interface. The new user interface is effective in two ways. 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 manipulation. Second, the overall execution time for accomplishing a given task (e.g. visiting a sequence of locations in the scene) 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 [10]. In its preprocessing step of this approach, the algorithm builds a roadmap that captures the connectivity of the free-space such as the one shown in Figure 1. At run time user mouse inputs are transformed into desired goal configurations. We then evoke the path planner to compute a collision-free path from the current configuration to the goal configuration whenever the two configurations cannot be directly connected with a straight-line path. 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 for execution. In summary, the system uses motion-planning techniques to voluntarily assist a user in navigation through difficult areas of a virtual scene.

3.2. Extensions to large-scale environments

The roadmap planner may not scale up well to deal with large-scale environments. The roadmap planner in our previous experiments is efficient for several reasons. First, we assume that the virtual world is a static bounded world 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 limited, and the time for searching the roadmap (an undirected graph) is insignificant.

The first factor is not directly affected by the scale of

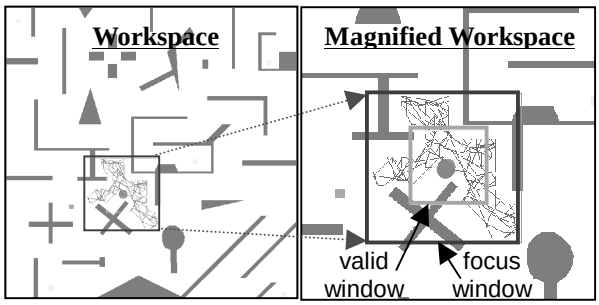

Figure2. RRT in the focus window

the world 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 update of roadmap

The key for extending the new interface to large virtual environments lie on the facts that the path-planning problem for our navigation control usually can be confined in a local region around the current viewpoint. No matter how large the whole world is we can expect that two consecutive configurations be close to each other under regular navigation operations. Thus, we can define a window called *focus window* in the workspace around the viewpoint 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 define another window, called *valid window*, which also moves with the viewpoint from time to time. Whenever the viewpoint exits the valid window we update the focus window and the valid window according to the current configuration. The size of the focus window determines the size of the roadmap while the size of the valid window determines the update frequency of the dynamic roadmap.

The challenge now becomes how to maintain the roadmap in the focus window as the viewpoint 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. Therefore, 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 type 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

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

4.1. The RRT data structure

Recently a new data structure called Randomized Rapidly-Exploring Random Tree (RRT) has been proposed as one form of roadmap for path planning.[12] 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. The tree consists of the initial configuration (x_{init}) only initially 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).

Two properties make the RRT algorithm attractive for our problem. First, the number of links amongst nodes is small since it uses a tree structure instead of a general interconnected graph in the traditional roadmap approach. As a result, the storage requirement and computation time for update a tree is also smaller. Second, the algorithm produces a tree that is typically a good representation of the freespace. It has been shown that the RRT approach tends to explore area that hasn't been explored yet [12]. The first property is crucial for our application since the size of the roadmap is limited by the physical memory we have as well as the time allowed to update the roadmap (which is typically much less than fractions of a second).

4.2. Updating RRT in the focus window

When the viewpoint 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 viewpoint 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. These differences consist of two parts: deletion and addition of nodes. For the example in Figure 4(b), as the viewpoint move northeast, we need to delete the nodes in the L-shape region D at the lower left corner and add new

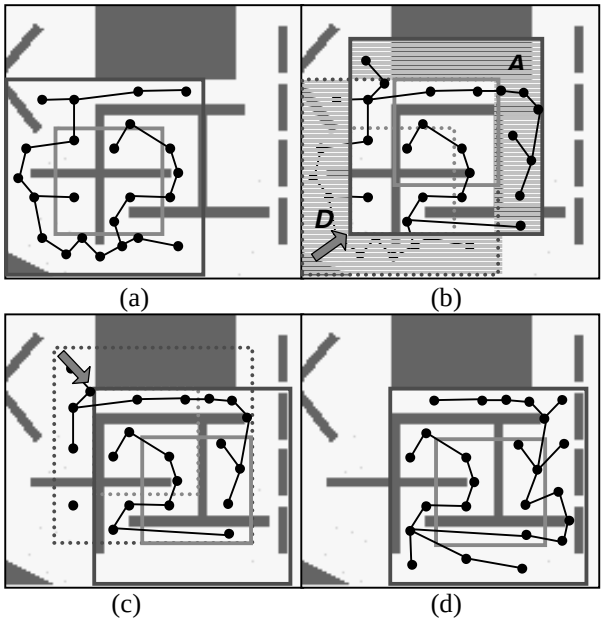

Figure 4. incrementally updating the RRT as the focus window moves

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 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 trees into a single tree. For example, in the RRT in Figure 4(c), the original tree is split into three subtrees. As the viewpoint 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. If two trees can be connected to the new node, these two trees are merged into one. In the merge operation, we also need to reverse the parent-children relation 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 the most fundamental and time-consuming routine in path planning. In our current implementation, we assume that the viewpoint 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

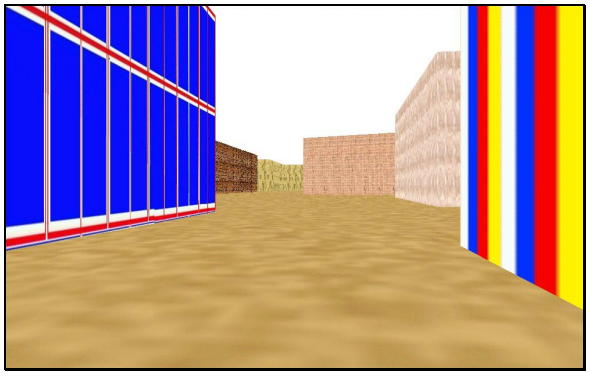

Figure 5. A snapshot of the VRML browser during the navigation experiments

maintain such a bitmap for the whole configuration space. Consequently, we must also update the bitmap in the focus window on the fly as the focus window moves. In order to update the bitmap for the new focus window, 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 virtual world.

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 the 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.[22] At the time of our implementation, this VRML browser does not support collision detection yet. Therefore, we have to enhance the browser with our implementation of collision detection routines. These collision checks are evoked in the viewpoint update routine 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. Although the nodes in RRT must be inside the focus window, the goal configuration for the viewpoint may fall outside the focus window. In order to facilitate collision checks for this case, we build the C-space bitmap for a larger window of 48x48 around the viewpoint.

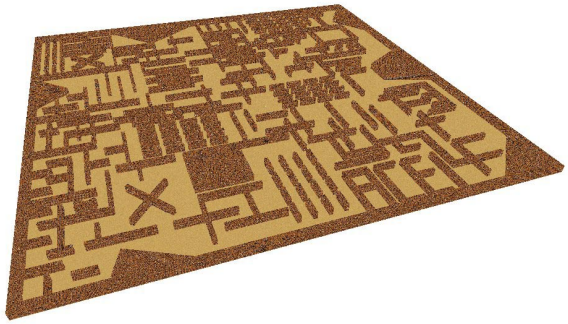

Figure 6. Top view of the large maze environment

6. Experiments

The goal of our experiments is to compare the performance of the new planner with dynamic roadmap update to that of the original planner with static roadmap. Since the benefits of incorporating planning into the navigation interface have 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 were carried out 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.

6.2. Experimental results

The experimental results are summarized 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.

From the data in Table 1, we have the following observations. The preprocessing time for P1 is significantly larger than P2 since it builds a complete roadmap for the entire C-space. In the navigation experiments with the same small world, 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 of P2 with the small world, the roadmap is updated 74 times

Table 1. Experimental data for planner performance for 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
ave. time for each path search request (ms)	275	50	83
ave. time for each path smoothing (ms)	186	49	81
no. of window updates	N/A	74	89
ave. time for each window update (ms)	N/A	41	22

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 dynamic roadmap (RRT) at different locations are shown in Figure 7. Experiments conducted in the third case (the large world) show that the average time for an update remains at the same degree as in the case of small world. If we examine the time for each update in more details, we can find that it consists of several portions: updating C-space bitmap, deleting nodes and adding nodes into the dynamic roadmap. 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 viewpoint. 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. The time spent in the planner consists of two parts: roadmap update and path query. First, the number of nodes in the dynamic roadmap does not change 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 been 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 with all obstacles at run time, the operation will depend 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

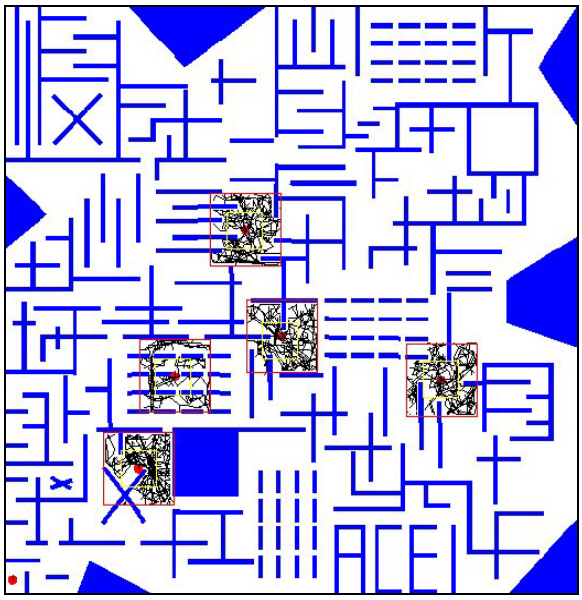

Figure 7. Examples of RRT's in the focus window at different times in a large virtual world

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 does not affect the performance until the world size becomes really large.

Although path planning is typically considered a time-consuming activity, 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 problem in a small window independent of the world size. On the other hand, the planner may fail to find a feasible path simply because the path cannot be entirely lie inside the window. Therefore, there exists a tradeoff between completeness and efficiency. The choice of the sizes for the valid window and the focus window in our experiments are determined empirically for now. It would be a subjective matter to determine their sizes, and they should actually vary for machines with different processing speeds.

The current computation bottleneck for our navigation application is actually on graphics rendering. The frame rate can easily become unacceptable for large worlds. This is why we did not do experiments on a world much larger than the current size. Therefore, in order to support visualization of large virtual worlds with architectural settings, we also need to consider the problem how to make good use of occlusion to speed up rendering. This is also an active research topic in computer graphics that is not addressed in this paper.

7. Conclusions and Future Work

In this paper, we have proposed improvements to our previous work on designing an intelligent navigation interface for architectural walkthrough applications. We extend the planner to consider the case of large virtual worlds by proposing a planner that maintains a special form of dynamic roadmap (RRT) at run time. This roadmap construction is confined in the focus window that is updated on demand as the viewpoint moves. Our experiments show that the planner has the same degree of computation efficiency as the previous planner with a static roadmap. This efficiency is also retained for large virtual worlds in our experiments. The benefits of using this type of intelligent user interface (mainly on overall execution time) can therefore be successfully extended to large virtual worlds. We also have analyzed the time complexity and report experimental data on various routines in path searches and roadmap updates. 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.

A main problem preventing us from doing experiments with even larger environments is on the speed of graphics rendering. In order to make our results be more applicable to large worlds, graphics rendering acceleration techniques accounting for large occlusions needs to be incorporated into the experimental 3D browser. In addition, we believe that the size of the valid window and focus window should be made adaptive to CPU processing speed as well as user navigation behavior.

Acknowledgments

This work was partially supported by grants from National Science Council under contact NSC 89-2218-E-004-001.

References

[1] J. Barraquand, L. Kavraki, J.C. Latombe, T.Y. Li, and P. Raghavan, “A Random Sampling Scheme for Path Planning,” in *International Journal of Robotics Research*, 16(6), P759-774, December, 1997.

[2] J. Barraquand and J. Latombe, “Robot Motion Planning: A Distributed Representation Approach,” *International Journal 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] Chen, Mountford, and Sellen, “A Study in Interactive 3D Rotation Using 2D Control Devices,” in *Proceedings of Computer Graphics* (*SIGGRAPH88*), 22(4):121-128 ,1988.

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

[6] P. K. Egbert, and S. H. Winkler, “Collision-Free Object Movement Using Vector Fields,” in *IEEE Computer Graphics and Applications*, 16(4):18-24, July, 1996.

[7] A. Hanson; E. Wernert, “Constrained 3D navigation with 2D controllers,” in *Proceedings of the 8th IEEE Visualization '97 Conference*, 1997.

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

[9] M.R. Jung, D. Paik, D. Kim, “A Camera Control Interface Based on the Visualization of Subspaces of the 6D Motion Space of the Camera,” in *Proceedings of IEEE Pacific Graphics’98*, 1998.

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

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

[12] J. Kuffner, and S. LaValle, “RRT-Connect: An Efficient Approach to Single-Query Path Planning,” in *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, May 2000.

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

[14] T.-Y. Li, L.K. Gan, and C.F. Su, “Generating Customizable Guided Tours for Networked Virtual Environment,” in *Proceedings of 1997 National Computer Symposium (NCS’97)*, Taichung. Dec.1997.

[15] T.-Y. Li, J.M. Lien, S.Y. Chiu, and T.H. Yu, “Automatically Generating Virtual Guided Tours,” in *Proceedings of the Computer Animation '99 Conference*, Geneva, Switzerland, pp99-106, May 1999.

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

[17] H. Lieberman, “Integrating User Interface Agents with Conventional Applications,” in *Proceedings of ACM Conference on Intelligent User Interfaces*, San Francisco, January 1998.

[18] M. Maybury and W. Wahster (eds), Readings in Intelligent User Interfaces, Morgan Kaufmann: Menlo Park, CA.

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

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

[21] D. Xiao, R. Hubbold, “Navigation Guided by Artificial Force Fields, “ in *Proceedings of the ACM CHI’98 Conference*, pp179-186, 1998.

[22] VRML97 International Standard, URL: http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[23] The Java3D and VRML working group, http://www.vrml.org/WorkingGroups/vrml-java