Appear in Proc. of 12th IPPR Conf. on Computer Vision, Graphics, and Image Processing, P525-531, Taipei, 1999.

An Intelligent User Interface for Architectural Walkthrough

Tsai-Yen Li (李蔡彥) and Hung-Kai Ting (丁竑愷)
Computer Science Department, National Chengchi University
64, Sec.2, Chih-Nan Road, Taipei, Taiwan 11623, ROC
e-mail: {li, s8420}@cs.nccu.edu.tw

Abstract

Due to the rapid evolution of graphics hardware, interactive 3D graphics is becoming popular on desktop personal computers. However, it remains a challenging task for a novice user equipped with a 2D mouse to navigate in an architectural environment efficiently. We think the problem is partly due to the fact that the level of navigation control that a user needs to provide is too low. In this paper, we propose a novel approach to improve the effectiveness and efficiency of the 3D user interface for architecture walkthrough. We adopt a path planner with probabilistic roadmap to help users avoid unnecessary maneuvers due to collisions with the environment. We modify a Java3D implementation of VRML browser to incorporate the path planner into the user interface. Experiments show that our implementation of path planner is very efficient and can be seamlessly incorporated into the navigation control loop. The overall navigation time for traversing a sequence of checkpoints in a maze-like environment can be improved by about a factor of two if the intelligent user interface is used.

1. Introduction

Traditional Virtual Reality (VR) often refers to immersive applications only. However, its definition has become broader with the introduction of the Virtual Reality Modeling Language (VRML) language[23] and the browsers supporting this format[21][22]. Typically, these browsers are designed to run on a regular desktop personal computer connected to the network. The developments of this 3D standard and 3D graphics acceleration hardware have greatly sped up the evolution of interactive 3D graphics on desktop PC's. While this form of VR, which we shall call *desktop VR*, is becoming prevalent, it is still a great challenge to design a good user interface for a novice user equipped only with a 2D mouse.

A typical VRML browser supports several navigation modes, such as WALK, FLY, EXAMINE, etc., and almost all browsers support the WALK mode for applications such as architectural walkthrough. Most of these browsers also support collision detection between the viewpoint and the environment to prevent the viewpoint from penetrating the obstacles and to increase the degree of realism. 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. It can neither move forward nor rotate at these locations without moving backward first. Users often feel frustrated with this kind of maneuvers especially when the frame rate is not high enough for smooth, responsive interactions.

We think the main problem is due to the facts that the level of navigation control that a user need to provide is too low, and the frame rate for complex scenes is still not high enough for precise control. There have been interesting philosophical debates on designing an intelligent user interface.[19] Direct manipulation has been shown to be an effective metaphor for interface design since the system behavior is more predictable than the intelligent user interfaces based on agent technologies. 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 because manipulating a complex virtual scene with a 2D mouse may not be efficient and definitely is not very intuitive for novice users. However, we think the mouse inputs can actually reflect the user intention for moving direction, but it is simply the problem that the user interface system is not smart enough to compute a collision-free motion to the destination automatically.

In this paper, we propose a novel approach of using efficient path planning algorithms in the control loop of 3D interactions to compute collision-free maneuver paths. We predict the locations where a user would like to move to from the mouse input and compute a collision-free path from the current configuration to a predicted goal configuration. These paths will then be followed by the viewpoint unless the user cancels the motion voluntarily. We implemented an efficient randomized roadmap planner that has been incorporate into the user interface of a common VRML browser. Our experiments carried out by users of various 3D experiences show that the overall navigation time can be significantly reduced with this intelligent user interface.

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 the path-planning algorithm with the randomized roadmap approach in Section 3. In Section 4 we present our approach to the problem of incorporating planning into the user interface control loop. We will 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 pertaining to our work fall into two categories: *3D user interface design* and *path planning*. Both problems require multidiscipline training for producing in-depth research results. Traditionally, the user interface design issues are addressed in the field of computer graphics and the path planning problem is studied mainly in Robotics.

2.1. 3D user interface design

User interface has been an indispensable component in a computer system since the time computers were invented. Many researches are undertaken to invent new efficient ways to communicate with a computer and on evaluating the effectiveness of these interfaces. Among these interfaces, being capable of interacting with virtual 3D environments has become a design trend for future user interfaces. VR-types of interfaces such as Head Mounted Display (HMD), 3D tracking devices, data gloves, force feedback joysticks, etc, are all good examples that are under active studies and development. New metaphors such as eyeball in hand, and flying vehicle in hand have been proposed and tested.[3] It is reported that most users like the idea of eyeball-in-hand metaphor in the context of virtual space exploration. However, the great challenge comes when we are asked to manipulate a 3D virtual scene only with a regular 2D mouse on a desktop computer. Some work has been carried out to design intuitive interfaces for controlling 3D rotations with 2D devices.[6][16]

Most of these proposals 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.[19] We think there will not be a clear-cut answer to this question. Instead, 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 take the wheel and have a full navigation control.

Although many intelligent user interfaces have been proposed in the literature, most of them are not for 3D manipulation.[13][15] 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 since the users should be allowed to concentrate on scene viewing instead of be distracted by low-level navigation controls. Li, et al. [11][12] also proposed an auto-navigation system capable of generating customized guided tour based on high-level user inputs. Kuffner [8][9] also utilizes fast path planning techniques to assist real-time animations. Other work also suggests using vector fields [5] or force fields [20] to guide animation. However, most of these approaches use geometric reasoning techniques as a tool for control delegation. They use a third-person view to specify the desired tasks, which is very different from the first-person view commonly used in the direct manipulation metaphor.

2.2. Path planning

The path planning problem (or the so-called *Piano Mover's Problem*) has been well studied in the past three decades. A good survey of path planning algorithms can be found in [10]. It has been shown to be a PSPACE-hard problem, and its computational complexity is exponential in the degrees of freedom (DOF) that the moving object has.[18] Due to the curse of dimensionality, 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]

In the past few years, a new path planning scheme called *random sampling scheme for path planning* is reported to be effective in solving practical problems in various applications with high dimensionalities.[1] A special version of planner with this random sampling scheme is called the *probabilistic roadmap* method.[7][17] It spends a significant amount of time to preprocess the connectivity information in the C-space such that it can answer path-planning queries afterward in a short amount of time. The type of planner is good for applications where static environments can be assumed and several planning queries are needed.

3. The Path Planner with Randomized Roadmap

In this paper, we propose to incorporate the path planning algorithms into the interface control loop to assist user navigation with direct manipulation (first-person view). Since the minimal frame rate for interactive 3D navigation is about 10fps (frame per second), it imposes a constraint on the acceptable time that one can spend on planning at each control cycle. This constraint will then limit the complexity of the planning problem for our system. Therefore, in this section we will describe the path-planning problem we consider with reasonable assumptions. We will also describe the randomized roadmap algorithm we adopt and explain why it is adequate for our application.

3.1. The path-planning problem

Instead of considering the general 3D interaction

problem, we only consider the application of architecture walkthrough. Almost all of the 3D browsers available in the public domain support this type of navigation mode. It is called the WALK mode in most VRML browsers. In architecture walkthrough, we can reasonably assume that the viewpoint stays on a horizontal plane. Therefore the virtual camera (representing the viewpoint) and the obstacles in the virtual environment can be reasonably represented by 2D polygons. The virtual camera can move freely in the plane and therefore possesses 3 DOF.

Many efficient path planners are reported to be able to compute a collision-free path in fractions of a second for environments of reasonable complexity. These planners fall into two categories: *one-shot* and *many-shot*. One-shot planners do not make assumptions about the environment and simply take the world description at run time. However, it might take a few seconds in the worst case for these planners to come up with a collision-free path. On the other hand, the many-shot planners, such as the randomized roadmap planner adopted in our system, may spend a reasonable amount of time initially in preprocessing the configuration space for future path-planning queries. These planners usually assume that the environment does not change frequently; otherwise, they will need to redo the preprocessing step whenever the environment changes. The planning times for these planners are better bounded since the planning problem can usually be reduced to only a graph search problem at run time. Therefore, this type of planners is more suitable for real-time user interactions.

3.2. The randomized roadmap planner

The path planner with the randomized roadmap approach, which we shall call the *roadmap planner*, belongs to the category of many-shot planners. It consists of two phases: *learning* phase and *query* phase. In the learning phase, the planner samples the C-space and builds a connectivity graph for the freespace (the set of collision-free configurations). Several strategies have been proposed in the robotics literature to perform the sampling. After enough configurations are sampled in the freespace, the planner will try to connect nearby configurations with a simple path computed by a local planner. The result is a connectivity graph capturing the topological structure of the freespace. An example of the roadmap is shown in Figure 1. This graph consists of 512 nodes connected in a 3D C-space. However, for clarity, the graph in Figure 1 is drawn directly in the 2D workspace by ignoring the orientation component of a configuration.

In the query phase, the planner is given a pair of configurations (the initial and goal configurations, denoted by q_i and q_g, respectively) and is asked to find a collision-free path connecting them. The roadmap planner will first try to connect q_i and q_g to any nodes, say q_i' and q_g', respectively, in the connectivity graph and then search the graph for a path connecting q_i' and q_g'. The path connecting q_i and q_g can then be constructed by concatenating path segments generated by the deterministic local planner. Since the graph search does not involve any expensive collision checks and the number of nodes in the graph is relatively small, the search time is usually quite small. A post-processing step is then applied to this path in order to produce a shorter and smoother path.

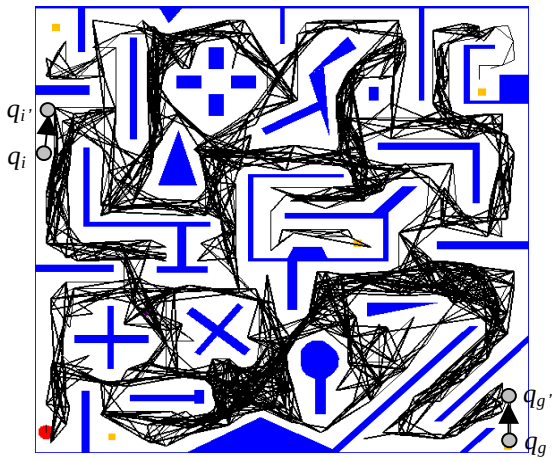

Figure 1: a sample probabilistic roadmap

There exist several empirical parameters that one can tune to produce good results efficiently. For example, the required number of sampled configurations might be different case by case. The more configurations are sampled and connected, the more time needs to be spent on the graph search, and also the more likely two configurations can be successfully connected. In addition, since the planner is probabilistic in nature, there could exist cases where the planner fails to find a feasible path that actually exists. However, for the user interface application we consider in this paper, we use the planner only as navigation assistance. Occasional failures do not cause fatal effects on the user interface. In fact, most of the planning problem instances encountered in this application are not very difficult. We would usually prefer an early failure instead of a long-waiting success since it might cause undesirable congestion in navigation control.

4. Intelligent 3D User Interface

4.1. The traditional user control loop

In an interactive 3D graphics program, such as a VRML browser, a user specifies his/her navigation commands through a 2D mouse. A typical program flow for a VRML browser consists of two threads: *input* and *animation*. The user-input thread is event driven while the animation thread is busy looping. A typical operation would require the user to drag a vector in the browser to represent the viewpoint velocity vector v. This velocity vector in the canvas space will be decomposed into a horizontal and a vertical component. The horizontal component, v_x, often refers to the rotational velocity while the vertical

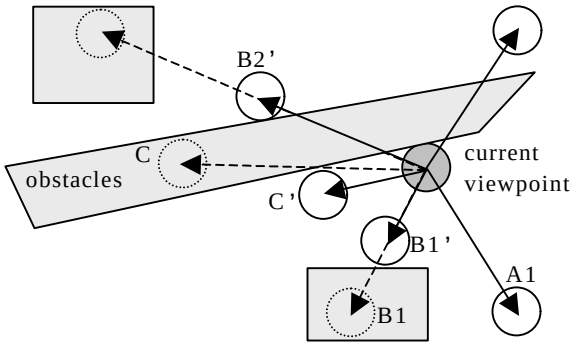

Figure 2: possible goal configurations and their modifications

component, v_y, often means the linear velocity along the forward direction. The input thread updates this motion vector whenever the mouse is dragged.

On the other hand, the animation thread loops indefinitely to perform configuration updates based on this velocity vector and then render the scene according to the updated viewpoint. In order to perform real-time navigation such that the distance traveled will not depend on the speed of the computer, the browser will multiply v_x and v_y by the time interval dt between two frame updates to obtain the displacement transformation T. The displacement transformation will then be multiplied to the current configuration q_i to obtain the next viewpoint configuration q_g ($q_g = T \times q_i$). Let d denote the translation vector between q_i and q_g. For browsers with collision-detection capability, potential collisions are checked along the line segment of d. If a potential collision might result from the movement, the viewpoint update will not be performed. This is the situation when the user gets stuck at a certain location. It is very often that the user has to move backward first in order to escape the trapping situation.

4.2. Predicting user intention

In our system, we propose to modify the animation loop such that the system will not give up movements in the situations where potential collisions might happen. Instead, the system will try to find a collision-free path for the viewpoint to follow whenever collisions are detected. We achieve this by maintaining a queue of collision-free configurations in the animation loop. Whenever the queue becomes empty, the system will try to fill the queue by generating a collision-free path according to the current velocity vector.

Now the problem becomes how the system predicts the intention of the user implied by the vector v. In other words, how does the system specify q_g in order to define an appropriate path-planning problem? There are three main cases to consider according to the legality of q_g and how it should be adjusted. These cases, depicted in Figure 2, are described as follows.

A. **No modification:** the projected q_g is legal, such as the A1 and A2 cases.

B. Direct modification: the projected q_g is illegal but it can be modified along d to become collision-free, such as the B1 and B2 cases. In our current system, q_g will be set to the first free configuration across the obstacle whenever possible (the B2 case). If not possible, it will be set to the farthest free configuration along d (the B1 case).

C. Indirect modification: the projected q_g is illegal but there exist no legal configurations along d (the C case). In this case, q_i already touches the obstacle boundary, and there are no legal q_g along the current forward direction. Therefore, q_g must be moved out of d. In our current system, we place q_g on the tangential component of d along the obstacle boundary.

By predicting the intention of the user, the system tries to move q_g to a nearby free configuration according to d. With these possible modifications, the path-planning problem for the user interface can then be clearly defined.

4.3. Computing smooth maneuver paths

After an appropriate goal is chosen, three types of results may be produced:

I. **Trivial path:** there exist no obstacles between q_i and q_g, and therefore a trivial straight-line path would be sufficient. For instance, cases A1, B1', and C' shown in Figure 2 all result in straight-line paths.

II. **Non-trivial path:** there is no straight-line path between q_i and q_g and therefore the path planner described in the previous section needs to be called to compute a feasible path. If a path is found, it will be smoothed and appropriately parameterized before putting to the configuration queue in the animation loop for execution.

III. **No path:** although both q_i and q_g are collision-free, the path planner fails to find a collision-free path connecting them. The configuration queue in the animation loop will remain empty, and no actions will be taken.

A user is allowed to intercept the execution of the path at any time by giving the system a cue such as releasing the mouse buttons or making a sharp turn. These cues should be consistent with conventions used in normal navigation operations. In the current system, when a path τ is generated, we also record the dragged vector v_t associated with τ. If the current vector v deviates from v_t for a certain threshold, the configuration queue in the animation loop will be flushed to empty, and the path is recomputed. Releasing the mouse buttons is treated as special case where v becomes null.

5. Implementation

5.1. Connecting to a VRML browser

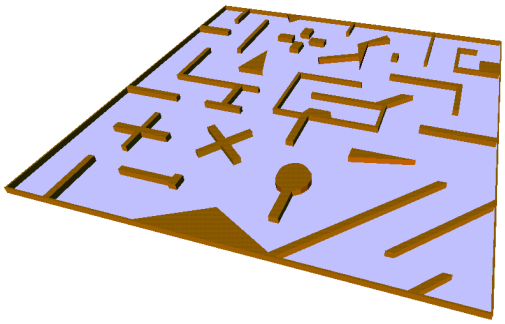

Figure 3: top view of the maze environment

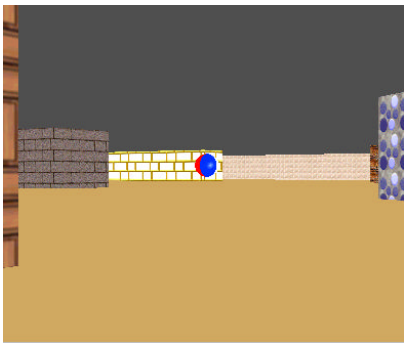

Figure 4: a snapshot of the VRML browser

In order to make the research result be more portable in the future, we choose to modify 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.[24] In these programs, we have mainly modified the routine for processing mouse event and the routine for updating the next viewpoint configuration. At the time of our implementation, this VRML browser does not support collision detection yet. Therefore, we have enhanced the browser with our implementation of collision detection routines. They are called in the viewpoint update routine to prevent potential collisions even when the path planner is not used.

5.2. The randomized roadmap planner

The roadmap planner has been implemented in the Java language. Two files are read by the system. The VRML browser reads in the VRML model of architectural environment, while the path planner reads in the corresponding 2D data file describing obstacle configurations. A separated maze editor has also been implemented to create both files consistently.

When the system starts up, we pre-compute the C-space obstacles with a well-known linear-time algorithm.[14] We store this information in a 3D bitmap of 128x128x128 for future collision detection lookups. In the learning phase of the roadmap planner, we perform a uniform sampling in the C-space. We uniformly divide the space into 8x8x8=512 regions and randomly sample up to four free configurations in each region. The system gives up sampling on a region after 20 trials. After the sampling step, the system will try to connect all pairs of nodes in the same or neighboring regions with collision-free straight-line paths. Therefore, there are up to 1024 interconnected nodes in the roadmap after the learning phase.

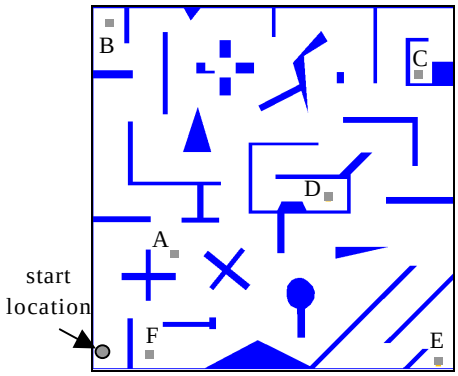

Figure 5: a 2D-layout map of the maze

In the query phase, we first connect q_i and q_g to some nodes q_i‘ and q_g‘ in the roadmap graph, respectively. Close nodes are tried first. After these starting and ending nodes in the graph are found, we use a modified A* algorithm to search for a path connecting these two nodes. If a feasible path in the graph is found, the straight-line segments along the graph path will then be concatenated to form the final geometric path. A smoothing routine will then be called to reduce the path length and increase its quality.

6. Experiments

Ten people were invited to test the implemented system. Eight of them are undergraduate students. Six of them major in Computer Science but only two use VRML browsers frequently. The other two testers do not use computers regularly. They are given a short instruction about how to use the browser and three minutes to warm up before the experiments start. In order to compare the effects of incorporating planning into the system, an experiment consists of two runs (with and without planning) of the same task sequence. In order to avoid unnecessary biases created by practice time, the run with planning is always done first.

6.1. Experimental settings

The experiments were carried out on a regular PC with a Celeron 300A processor. The experimental scenario consists of six checkpoints in a maze-like environment for a user to visit sequentially. A top view of this environment is shown in Figure 3. The user navigates in the maze with a first-person view and the WALK mode provided by the VRML browser. A bouncing ball is used to help the users identify the targeting checkpoint. An example of the rendered 3D scene is shown in Figure 4. In order to assist the user in finding the next checkpoint, we also provide a 2D-layout map, as shown in Figure 5, at the side of

	w/ planning	w/o planning
total execution time(sec)	243	421
no of navigation steps	2498	3941
preprocessing time (sec)	2.5	0
avg. time for computing the next step (ms)	18.9	7.4

Table 1: comparison of walkthrough efficiency with or without path planning

	n_1	n_2	n_3	n_4	n_5	t_1	t_2	t_3
u1	746	1644	888	30	19	2.01	129.4	281.5
u2	790	1592	773	36	17	1.44	124.9	287.5
u3	908	1653	736	65	50	3.05	209.2	471.4
u4	1476	2592	1117	87	63	3.15	220.5	427.7
u5	2737	3106	361	36	13	0.98	239.9	332.1
u6	2451	3070	640	36	14	1.30	261.4	335.5
u7	2091	2430	295	72	84	3.39	249.4	332.1
u8	1430	2419	972	59	39	2.66	238.1	585.2
u9	1861	2988	1024	104	93	4.25	370.9	514.8
u10	2942	3486	523	50	23	1.61	384.9	607.4
avg.	1743	2498	733	58	42	2.38	242.9	420.7

n_1: no of generated paths n_2: no of generated steps
n_3: no of planner generated steps n_4: no of planning called
n_5: no of paths being cancelled during their executions
t_1: total path-planning time (sec)
t_2: total execution time with planning (sec)
t_3: total execution time without planning (sec)

Table 2: statistic data of ten users using the intelligent interface

the VRML browser window. In each experiment, a user has to move the viewpoint from checkpoint A through checkpoint F as quickly as possible.

6.2. Experimental results

The experimental results consist of two parts: objective statistic data and subjective user comments. The comparisons of the system with and without planning are summarized in Table 1. The numbers are based on the average of the ten testers. The overall times taken to complete the requested task are 243 and 421 seconds, respectively, for the system run with and without planning. The performance speedup for the interface with planning is about 73%. In term of movement steps, about one third of the steps are saved after the planner is used. The cost to pay is that the system has to spend about 2.5 seconds in total to preprocess a given environment and about an average of additional 11.5 ms in each step to determine the next viewpoint configuration. (It still takes some time for the interface without planning to detect collisions.) The overall extra time spent on planning only consists of 6.21% of the overall execution time.

The subjective user feedback for the user interface with planning is positive in general. Some of the users may find the viewpoint hard to control in the beginning (especially for those who like to play games with keyboards) but all of them can get used to the control very soon. Once they are in a good command of the interface, navigation efficiency is greatly improved. In summary, they all regard the path-planning capability

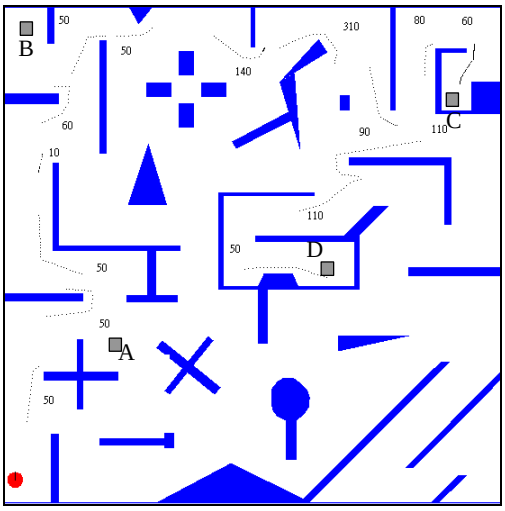

Figure 6: planner generated non-trivial paths

as a very considerate and desirable feature for a 3D VRML browser.

6.3. Analysis

In order to understand how path planning is used in our walkthrough experiments, we take a closer look at the statistic data collected from the ten experiments, as listed in Table 2. The number n_1 in the second column is the number of paths (including type I and type II paths in subsection 4.3) and n_2 is the number of steps generated for execution during an experiment. Although the number of calls to the path planner (n_4) is not very high, total number of steps (n_3) in these nontrivial paths constitutes a great portion (about one third) of n_2. Typical paths generated by the planner, depicted as sequences of dots, are shown in Figure 6. The numbers beside the paths are their lengths. During the navigation, some paths (trivial or nontrivial) (n_5) were cancelled due to significant deviation from the original user intention. The total time spent on path searching in the roadmap planner (t_1) is only a small portion of the overall execution time (t_2). Compared to the execution time for the interface without planning (t_3), the intelligent user interface with planning indeed results in very consistent and significant improvements.

7. Conclusions and Future Work

In this paper, we have proposed a novel approach to design an intelligent user interface for architectural walkthrough applications. A path planner with a randomized roadmap approach is used to assist users in navigating through difficult areas where the users often get stuck with the traditional user interface. This planner has been successfully integrated with the low-level control loop in a VRML browser. Our preliminary experimental results show that, with the help of the planner, the overall navigation time can be consistently and significantly saved. We believe that this

intelligent user interface is effective because it delegate some of the geometric reasoning tasks to the computer while retaining the advantages of direct manipulation.

There exist limitations on our current implementation. For example, we currently assume a static and bounded workspace in our path planner to facilitate roadmap construction. However, it is more desirable (for path planning in general) to be able to consider dynamic and unbounded workspace possibly with incremental roadmap construction. The effect of path planning on the walkthrough type of 3D interface design deserves further studies. For example, more experiments need to be carried out for different tasks, on different virtual scenes, and on different systems of various computing powers. We believe that the effects will even more significant for complex scenes on slower machines.

Acknowledgments

This work was partially supported by grants from National Science Council under contacts NSC 88-2815-C-004-001-E and NSC 88-2218-E-004-002.

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] Chen, Mountford, and Sellen, “A Study in Interactive 3D Rotation Using 2D Control Devices,” *Computer Graphics*, 22(4):121-128 ,1988.

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

[5] 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.

[6] 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.

[7] 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.

[8] J.J. Kuffner. "Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control". In *Proceedings of CAPTECH '98: Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, Geneva, Switzerland, Nov. 26-28, 1998.

[9] 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.

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

[11] 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.

[12] 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.

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

[14] T. Lozano-Perez, “Spatial Planning: A Configuration Space Approach,” in *IEEE Transactions on Computers*, 32(2):108-120, 1983.

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

[16] Neilson and Olsen, “Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices,” in *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pp 175-182, 1986.

[17] M. Overmars, and P. Svestka, “A Probabilistic Learning Approach to Motion Planning,” in the *Proceedings of the 1994 Workshop on the Algorithmic Foundations of Robotics*, pp19-37, 1994.

[18] 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.

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

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

[21] WorldView VRML browser, URL: http://www.intervista.com/

[22] CosmoPlayer VRML browser, URL: http://www.cosmosoftware.com/

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

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