為網路虛擬實境產生自動導覽路徑
Generating Customizable Guided Tours For Networked Virtual Environments

李蔡彥 甘連凱 蘇建富
Tsai-Yen Li, Lien-Kai Gan, and Chien-Fu Su

國立政治大學資訊科學系
Computer Science Department
National Chengchi University, Taipei, Taiwan
{li,s8204,s8237}@cs.nccu.edu.tw

摘要

以3D互動式電腦繪圖的方式，在網際網路上提供資訊已日趨普遍，但現今大部分的網路虛擬實境系統，對一般以2D滑鼠操控電腦的使用者而言，仍不十分方便使用。改進此類系統的方式之一，在於為使用者於瀏覽虛擬環境時提供一些導引。在本文中，我們提出一個軟體系統，能於網路虛擬環境中，自動產生個人化的導覽路線。此一已實做完成的軟體，整合了機器人學中的運動計畫演算法，及網路虛擬實境規格的最新發展（例如VRML2.0）。使用者只需以滑鼠點選所欲前往的位置及參觀模式，此系統便能自動計算出參觀路徑，並於網路虛擬實境系統中引導使用者前往瀏覽這些參觀點。實驗結果是以本系大樓3D模型為例做展示。我們預期此研究結果可被用於各博物館和國家公園的導覽系統中，為當地或遠端使用者提供新的瀏覽協助。

關鍵字：自動導覽、路徑計畫、網路虛擬實境、VRML虛擬實境建模語言、互動3D圖學

Abstract

Interactive 3D graphics is becoming a popular way of publishing information on the Internet. However, most networked VR systems today are still not very intuitive to use, especially for a novice user equipped with a regular PC and a 2D mouse. One way to improve such systems is by providing users with some guidance during the navigation of a virtual environment. In this paper, we present a software system[1] capable of generating customizable guided tours for a networked virtual environment. The implemented system incorporates motion planning algorithms in Robotics into the latest development of network-based virtual reality standards such as VRML 2.0. A user can simply click on the locations they wish to visit and select a visiting sequence, and the system will automatically generate a tour plan and guide the user walk through the desired locations. Experimental results are presented using a 3D model of our departmental building. We expect the result of this research to be used as new navigation assistance to local or remote visitors of docentry systems located in museums or national parks.

Keywords: Guided Tours, Path Planning, Networked Virtual Reality, VRML, and Interactive 3D Graphics

1. Introduction

Computer technologies have been widely used to provide touring information to visitors. Most tour-guiding systems existing today display information only in text or in static pictures. A few systems can support multimedia contents such as video and audio clips, but very few can display interactive 3D graphics. This kind of 3D-display capability used to be the privilege of expensive high-end workstations in Virtual Reality (VR) researches. However, as personal computers become more powerful and affordable, displaying interactive 3D graphics on a regular PC is no longer a problem. In addition, as computer network becomes prevalent, downloading a virtual scene and visiting it from a distance becomes possible or even popular. For example, with the development of Virtual Reality Modeling Language (VRML), one can now download and visit a virtual scene through a regular web browser.

However, browsing experiences are usually far from perfect for novice users equipped with regular personal computers. First, it is not intuitive for them to use a 2D mouse to manipulate 3D scenes. Second, the complexity of the models that their PC's can handle is usually quite limited. It is true that when the VR-related industry becomes maturer, 3D peripheral devices might become more accessible in the future. However, it is likely that computing power will never catch up the increasing user demands for high-quality visual experiences. We believe that the key to promoting interactive 3D graphics on the Internet lies on the ability to provide some kind of intelligent assistance to the users of a networked VR system.

In this paper, we present a software system that can generate customizable guided tours for users in a virtual environment. This system incorporates motion planning algorithms in Robotics into the latest developments in networked VR. A 3D model of our departmental building in VRML was built to demonstrate features of our

[1] This work was partially supported by NSC under contract No. NSC86-2213-E-004-006.

system. A user can simply click on the locations that they wish to visit in a 2D-layout map and select a traversing mode, the system will automatically generate a tour plan and guide the user through these locations in the virtual building. In addition, when desires, the user can intervene the system during the tour with default browser commands or additional functions provided by the system.

The rest of the paper is organized as follows: In Section 2 we describe the researches in current tour-guiding systems, VRML, and motion planning pertaining to our work. In Section 3 we describe the problems that we encountered in developing a tour planning system and some necessary simplifications for making the problems more tractable. We then present an overview of our system as well as its constituent modules in Section 4. In Section 5, we give a more detail account on a few implementation issues of the software system that we have developed. In Section 6, we present sample snapshots of our graphical user interface and some experimental results on system performance. We conclude our paper in Section 7.

2. Related Work

2.1 Guided tours and VRML

Surveys on the functions of current guided-touring systems in popular museums and parks show that most visitors use these systems to acquire site information such as how to visit all popular points without wasting time wandering around. They also conclude that interactive multimedia systems will be a trend in developing the next generation of touring systems [6]. Virtual Reality is a candidate technology that can add realistic visualization into such systems. However, VR systems usually require vast investment on expensive peripheral devices to achieve realistic effects. By sacrificing certain realistic effects and focusing on what regular PC's can do, it is still possible to bring exciting visualization to general users. However, to really popularize VR technologies, a portable data format over different platforms should also be agreed upon. This is especially important for bringing 3D graphics onto the Internet.

VRML is an example of such platform independent standards. After the VRML 1.0 specification was released, many *Virtual Museums* on the web were built in this format [12]. However, because this version of specification only defines the geometric modeling aspect of 3D computer graphics, virtual scenes created using this specification can only contain static objects. Several addenda were proposed to add to it more dynamic capabilities such as multimedia and animation supports. Among these proposals, the "Moving World" proposal is the one that becomes the basis of the VRML 2.0 specification [3]. With this new specification, the number of dynamic applications that can be accomplished by networked VR is increasing rapidly [13].

However, not every desirable feature was included in the final specification of this version. For example, no agreements were reached on how to classify and describe object behaviors in a virtual environment. Another missing feature is the ability to control a virtual scene through external programs written in other programming languages. The External Authoring Interface (EAI) addendum to VRML 2.0 was proposed to fill this need [10]. Currently, several popular VRML browsers have claimed to support this interface as part of their standard functions [14]. With this addendum, it becomes easier to add complex programming logic into a VRML world.

2.2 Motion planning

It is highly desirable for a VR system to generate guided tours automatically for its users. These tour plans should be able to take a user through all desired locations without colliding with obstacles in the environment. The problem of planning such a collision-free path for a moving object is the so-called "Piano Mover's Problem" in the Robotics literature. Researches have shown that the general path planning problem is PSPACE-hard, and its complexity grows exponentially in the number of degrees of freedom (DOF) that a moving object has. Nevertheless, several researches have reported efficient algorithms that can solve difficult cases in reasonable time [2][4]. The philosophy behind these researches is pragmatism. Although the problem is theoretically difficult, it is possible to develop algorithms that run efficiently most of time on most real-life examples.

The tour-planning problem we have is very similar to the problem of generating a navigation plan for an autonomous robot in a known environment [7]. However, some differences make the design of our system challenging. For example, the length of the overall tour generated for human visitors are typically longer. No uncertainties need to be considered like in the case of robot navigation. In order to provide prompt responses to interacting users, the planning time of our system needs to be as short as possible. Furthermore, we have to integrate planning results with the VRML standard in order to take advantage of latest developments in interactive 3D graphics.

3. Problem Description and Simplifications

3.1 The planning problem

Given a set of user-selected visiting points, the system is to solve an undetermined sequence of path planning problems for traversing these locations. There are two fundamental subproblems that we have to address: how to find a collision-free path from one location to another, and how to find an optimal sequence for traversing these locations. The first subproblem is a typical path planning problem in Robotics while the second subproblem is an NP-complete Travelling Salesperson Problem (TSP). Both subproblems are known to be intractable. However, we will show that with appropriate

simplifications and good heuristics, these problems can be solved in reasonable time in practice.

We first define some terminology that we will use in later sections. A *configuration q* of an object moving in a 2D workspace (*WS*) can be modeled with three parameters: (x, y, θ), representing the position and orientation of the moving object. A *Configuration Space* (*Cspace*) is defined over $R^2 x[0,2\pi)$ to represent all possible configurations of the moving object. The open subset of a Cspace containing all collision-free configurations is called *freespace*. A *feasible path* is defined as a sequence of configurations in freespace connecting the given initial and the goal configurations.

The first subproblem we have is a basic path-planning problem for a 3-DOF object moving in a 2D workspace. Several researches in Robotics have reported complete algorithms that can solve a problem of this complexity in a few seconds [2]. However, a typical guided tour contains several instances of this basic problem. The time spent in planning could add up very quickly. To ensure that the overall planning time is acceptable in an interactive environment, appropriate simplifications are necessary.

The second subproblem is to find a sequence of paths connecting the user-selected locations such that the overall travelling distance is minimized. This is similar to the TSP problem in graph theories and is known to be NP-complete. For a TSP problem with Euclidean distances between locations, there are approximate algorithms that can find near optimal solutions in polynomial time [5]. However, for a workspace containing obstacles, the real distances between these locations are unknown prior to the search process. One straightforward way to solve this problem is to compute the shortest path for every pair of locations and apply an approximate algorithm to the TSP problem. However, as one can predict, this two-step approach will not yield satisfactory performance.

3.2 Simplifications

Simplifications are necessary for making our problem more tractable. First, we try to simplify the geometric model of the moving object. In our case, navigation happens in a structured environment (for example, in a building), it is reasonable to assume that the geometry of a human visitor be enclosed by a circle. Under this assumption, the parameters of a configuration in a path are decoupled into two parts: position (x, y) and orientation (θ). The position part can be computed with an efficient path planning algorithm, and the orientation part is added afterward to ensure the continuity of viewer orientation in making turns. The number of Cspace dimensions used in the planner can then be reduced to two, and the collision detection methods can be made more efficient as well.

Second, we assume that the distance between two locations can be approximated by the travelling distance

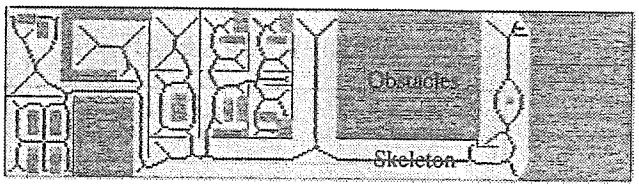


Figure 1: An example of skeleton for a freespace

on the skeleton of a freespace. An example of skeletons generated by the system is shown in Figure 1. This kind of skeleton is a discrete form of the Voronoi Diagram encapsulating roadmap information of a freespace. A location must connect to this skeleton (like connecting to a highway via a ramp) before it can be connected to another location in the workspace. A path found under this assumption needs to be smoothed afterward. The resulting smoothed path may deviate from the skeleton but the topology remains the same. For the problem of computing a guided tour inside a building, this is a reasonable assumption since we are not likely to guide the user through a narrow passage in a cluttered environment. Under this assumption, the size of search space for a collision-free path can be considerably reduced.

In addition, we also assume that the size of workspace and the locations of the obstacles inside do not change once the data is loaded into the system. The reason for making these assumptions is to simplify our implementation such that we do not have to deal with time-related issues during planning. In fact, it is a more challenging problem if the objects in the workspace can be changed on-line.

4. System Design

Our system consists of three major modules: a *3D-display* module, a *planning* module, and an *integration* module. The 3D-display module includes a VRML 2.0 browser and a data file. This data file contains inline programming logic as well as 3D geometric models that can be displayed alone in a VRML-enabled browser. The planning module has its own graphical user interface and provides path-planning services to the users. It could also be a standalone program that simulates the planned tour via its own interface. The integration module is built to facilitate data communication between the other two modules.

4.1 3D-display module

A common problem in viewing virtual scenes is that the size of a visible scene is limited by the size of geometry data a computer can handle. The quality of a scene often needs to be traded off with data size in order to display a large scene. If a display system can support automatic and smooth transitions between adjacent scenes, we can decompose a large scene into several smaller ones and display them on demand. In our 3D-display module, we make use of new animation features in VRML 2.0 to achieve automatic scene changes. Spe-

cifically, we use proximity sensors and hyperlinks at appropriate entrances and exits of a scene to trigger the loading of another scene.

Scene changes can be handled in VRML built-in nodes, but more sophisticated functionality needs to be accomplished by an external program. Indeed, VRML was designed to be a modeling language instead of a programming language. An interface that allows external programs to request or change the data in VRML models is crucial for our system. For example, in the 3D-display module we allow a user to navigate a 3D scene through default browser commands. Thus, in order to synchronize the current location of the viewpoint between the 3D-display and planning modules, we need to be able to get and set transformation data in a VRML model. There are two ways to achieve this in VRML 2.0: using a script node to connect a VRML browser to a Java program, or using EAI functions to connect it to a Java applet. We chose the EAI functions to control viewpoint changes in our system since it makes the integration of the VRML and the planning module easier as explained in later subsections.

4.2 Planning module

The input to the planning module include a 2D layout of the workspace, the current location of the moving object (visitor viewpoint in our case), a list of locations to be traversed, and a user selected traversing mode. This module outputs a smooth path going though all the desired locations. In order to produce such a path, the module has to prepare an appropriate search space (a skeleton in freespace under our assumption) and then apply a search algorithm to find a desired sequence and its associated path.

Since the moving object is modeled as an enclosing circle, the Cspace for a given workspace can be easily computed by expanding the workspace obstacles with the radius of the enclosing circle [8]. The resulting freespace is then used to compute a skeleton. We adapt the SKELETON procedure used in computing the "NF2" numerical potential fields [7] to construct the skeleton. The idea of this procedure is to incrementally expand the boundaries of obstacles in the Cspace in a way similar to moving a wave outward to the freespace. When the fronts of wave expansions from different origins meet, the touched portion forms the basic skeleton. The locations selected by a user may not be on the skeleton. Therefore, after the basic skeleton is built, we have to connect each selected location to a closest point in the skeleton. These connecting segments are treated as part of the skeleton that is used to search for a feasible path.

There are two basic traversing modes for a user to select: in-order and non-order. In the in-order mode, the system traverses locations according to the order that the user selects them. In the non-order mode, no particular order is imposed. The user lets the system find the sequence yielding a path that is as short as possible. The first mode is the easier case since it can be computed as

```
procedure GreedySearch(q_i)
    S ← all selected locations
    q_c ← q_i
    while S is not empty do
        q_new ← NearestConfig(q_c, S)
        remove q_new from S
        q_c ← q_new
    end while
end procedure
```

Figure 2: Greedy algorithm for path search

a concatenation of subpaths, each of which connects two predetermined locations. The problem with the second mode is more difficult as explained earlier.

For the second mode, we use a greedy algorithm to find a near-optimal tour plan. The psudocode for this greedy algorithm is shown in Figure 2. Initially every unvisited location is included in a candidate set S. We start from the current location q_c of the moving object and try to find the nearest location from q_c using a procedure called NearestConfig. This procedure takes q_c and S as arguments and performs a breadth-first search from q_c along all possible branches of the roadmap until an unvisited location in S is reached. This nearest location q_{new} is then removed from S, and its value is assigned to q_c. The same process repeats until S is exhausted, which means that we are done with visiting all locations. The final tour plan is a path consisting of the found subpaths concatenated in the order that they are discovered.

The path found in the previous step consists of a sequence of viewpoint locations, which are only the translation components of configurations in the path. We still need to determine the orientation of each configuration along the path. In our system, these orientation parameters are calculated based on the direction of the camera motion. We assume that by default the camera always faces toward the moving direction. The path found in the previous step may have abrupt directional changes along the path. At these points, we insert into the path some extra intermediate configurations that change gradually only in orientation.

The above greedy algorithm does not produce an optimal solution in general. First, the lengths of the paths on the skeleton may not be the final travelling distances after the paths are smoothed. Second, the skeleton is a general undirected graph that may contain cycles. When it does, the sequence found by the greedy algorithm may not be the optimal one. Nevertheless, the results produced by our system are reasonably well. It usually generates a path keeping a safe distance from the obstacles. This kind of paths is usually preferable to the real shortest paths.

4.3 Integration module

The overall structure of the system is shown in Figure 3. The 3D-display module, which includes an em-

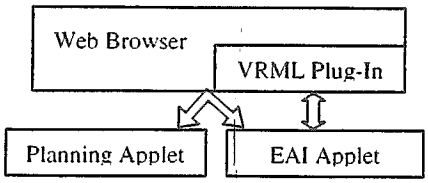


Figure 3: Overall System Structure

bedded VRML browser, is connected to a Java applet through the EAI package. This applet in turns communicates through the web browser with another Java applet implementing the planning module. When a scene is loaded, the EAI applet first registers itself with the planning applet. When the user asks for a tour plan, the planning applet queries the current viewpoint location through the EAI applet and then start planning. After a tour pat is found, the EAI applet starts to query new locations from the planning applet. The locations are retrieved one at a time until the end of the tour.

Alternatively, the planning module can be combined with this integration module and communication with the VRML world directly. However, we have chosen to separate them due to the following two reasons. First, logically the system should have two independent processing loops: one for updating the VRML world and the other for processing its own user interface. Second, each module can be tested separately without affecting the other. It would also be easier to replace any module with a different implementation in the future.

5. Implementation

In this section we describe in more details how some important constituent components of the above modules are implemented.

5.1 VRML models

We use our departmental building as a demonstrative example for our system. The geometric models in this example were created using commercial 3D modeling software such as 3D Studio. These models were then exported to files in VRML 2.0 format. All models were carefully designed such that the complexity of a scene is reduced in order to take advantage of potential optimizations used by common VRML browsers. Some animation functions are added manually into these geometric model files via VRML2.0 built-in nodes. For example, the transitions between different model files are achieved by placing ProximitySensor nodes in some predefined regions. Entering these regions will trigger an event that loads an appropriate scene into the browser.

5.2 Path planning

The path planner in the planning module computes a guided tour in three steps: a preprocessing step for a workspace, a path searching step for a given set of locations, and a postprocessing step for smoothing a found path.

The initial input to the planning module is a data file containing a 2D-layout description of the 3D scene in the display module. This layout description includes the size of the workspace and polygons representing walls and obstacles in the workspace. In the current implementation, this file is created manually. However, ideally this 2D layout should be generated automatically from its corresponding 3D scene by taking horizontal cross sections of the VRML models. This function should be implemented in future versions of the system.

Since the environment does not change once it is loaded into the system, we spend some time in the precomputation step to compute data pertaining to the workspace only. These data include the Cspace and the skeleton of the freespace. By doing so, we can save time when a user make more than one request in the same environment. In our implementation, the workspace and Cspace are all represented by a bitmap containing a grid in a resolution of 280 by 75. Grid cells occupied by obstacles are filled in the workspace using a scanline algorithm in 2D graphics. To transform this workspace to a Cspace, we use the wave-front algorithm in [1] to expand the boundary cells of the obstacle outward by the radius of the enclosing circle. This Cspace can then be used to look up collisions during the planning process.

The basic skeleton for a freespace is computed using the algorithm described in the previous section. This skeleton is stored as discrete points in another bitmap of the same dimension as the Cspace. For each planning instance, this basic skeleton is duplicated and extended to the user-selected locations. The resulting skeleton is then used in searching for a tour plan.

A breadth-first algorithm is used in searching for a feasible path. We start from the node containing the initial location and visit its neighbor nodes that are also part of the skeleton. These neighbors are stored in a list. Each of them will be used to generate the list of one step deeper/farther until the goal node is reached. Each node keeps a reference to its parent node so that we can recover the path by backtracking from the goal node to the initial node.

The paths generated by the planner are accurate up to certain resolution specified by the system. For example, the translation resolution in our system is less than or equal to 1/280 of the longest side of the workspace. The rotational resolution is less than five degrees. The planner will not ignore any passages that are wider than a unit cell and the generated path will not cause the moving object to jump over obstacles that are wider than a unit cell as well. The translation resolution of a path is ensured in both the path searching and smoothing processes while the orientation resolution is abide by in the postprocessing step. According to the frame rate of the display module, the system can re-parameterize the path in order to animate it with an appropriate speed.

5.3 Integration

In our system, the VRML module is a plug-in of a

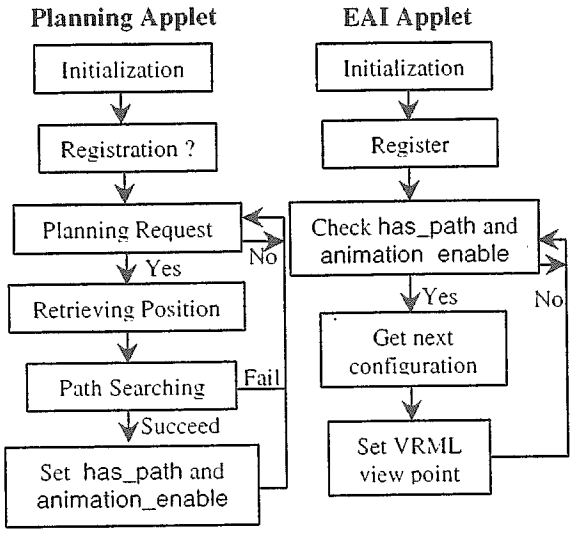


Figure 4: Flow charts for the main loops of the planning and EAI applets

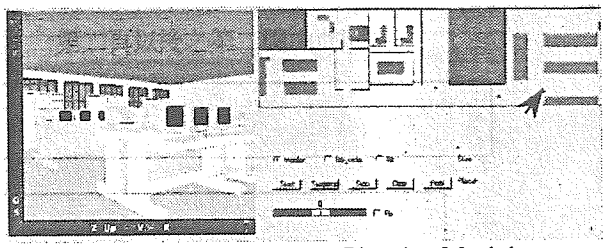
web browser while the EAI components of the integration module and the planning module are Java applets. The flow charts for the main loops in these two applets are shown in Figure 4. When the web page is loaded, the EAI applet first registers with the planning applet by calling the object model's API functions provided by the hosting web browser. Specifically, the `getAppletContext` function call is used by the EAI applet to get a handle to the named planning applet. It then calls a function provided by the planning applet to register itself.

Two flags in the EAI applet named `has_path` and `animation_enable`, are controlled by the planning applet to set animation status. These two flags are set to `TRUE` when a path is found. The `animation_enable` flag is set to `FALSE` when a user suspends or stops the animation. The `has_path` flag is set to `FALSE` by the planning module at the end of the tour. The EAI applet retrieves a new configuration for the next step of animation only if both flags are `TRUE`. The EAI applet then uses this new configuration to update the ViewPoint node in the VRML model.

6. Experimental Results[2]

6.1 User interface

The 3D-display and planning modules present their graphical user interfaces in different regions of the same web page. A snapshot of the interface showing that the system is guiding the user through the fourth floor of our department is shown in Figure 5. On the left of the page is a popular VRML plug-in called WorldView[14], and on the right is a Java applet showing the 2D-layout map of the floor presented by the planning module. To select a location, the user can simply click on the map. The planning module will automatically check if the location



3D-Display Module Planning Module

Figure 5: A snapshot of the system's user interface

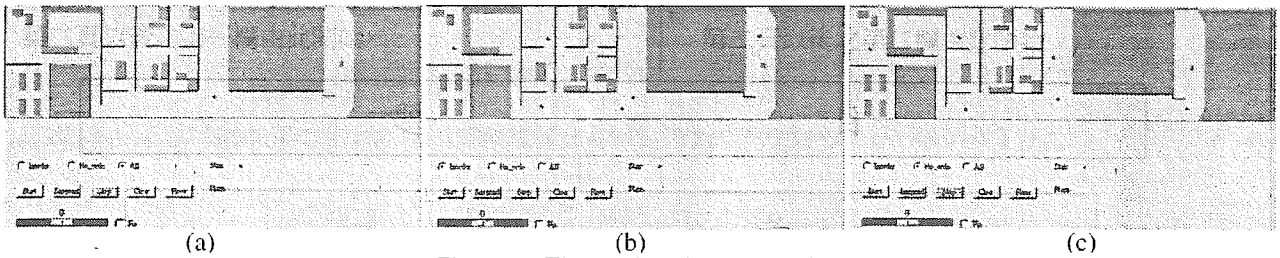
is collision-free before accepting it. When the animation of a guided tour is in progress, both interfaces reflect the current locations of the viewer. The planning applet also provides an interface for the user to set planning options and to control the animation. For example, the system allows the user to visit default or selected locations in the scene in the order they pick or in no particular order. The user can also suspend or stop the tour at any time during the animation. There is a command button that can generate a path that takes the visitor to the stair to a different floor. The system also allows the user to change viewing angles while keeping the same moving direction. This option is to simulate another degree of freedom of a human visitor: neck rotation. This rotation angle is added to a configuration by the planning applet before sending it to the 3D-display module.

6.2 Performance

The performance of the overall system mainly depends on the performance of the 3D-display and the planning modules. The VRML browser we used is considered to be one of the most efficient implementations of the VRML 2.0 specification. However, for models with complexity as in the example shown in Figure 5 (about 1.3MB in size), the system's performance is sluggish on a Pentium 133 PC with 64 MB of RAM. The frame rate is about 0.5 to one frames per second. The long delays between animation frames make it unacceptable to navigate the scene interactively using a 2D mouse. This was one of the main reasons that motivated us to add planning capability to the system to help the user navigate a 3D scene through automatically generated tours.

The performance of the planning module is also crucial for a user to accept the system. The user may lose his/her patience if he/she needs to wait for minutes for a customized guided tour to be generated. We have chosen Java as the programming language to implement our planner since currently it is the only language supported in EAI. Fortunately, our system still can generate a guided tour in a few seconds for visiting a typical number of locations. Figures 6(a)-(c) show examples for the third floor of our departmental building in three different traversing modes. The times for running these examples were measured on a Pentium-120 PC with 32MB of RAM. The preprocessing time used to initialize the planning module for this scene is 2.580 sec. In Figure

[2] The implemented system is available at http://hamm.cs.nccu.edu.tw/hamm/project/AutoTouring



(a) (b) (c)

Figure 6: Three planning examples

6(a), the planning time for visiting all default locations in no particular order is 0.71 sec. Figure 6(b) shows the case where the user would like to visit five locations in the order that he/she picked, and the planning time is 3.18 sec. Figure 6(c) shows the third case where the user selects four locations and does not care about the traversing order. The running time is 2.31 sec.

From these running times, we can get an idea about the performance of the planning system. However, further comparisons may not bear significant meaning. First, the running times depend on the number of selected locations, their actual locations, and the order that they are selected. It is not easy to quantify the effects of these factors. Second, since animation usually takes much more time than planning, what actually matters for the overall system is the length of the generated path. However, the total display time depends not only on the total length of the path but also on the frame rate at which the computer can display. On the other hand, the frame rate should be tunable to avoid rapid or abrupt viewpoint changes during the animation. Consequently, the performance of the system is more a subjective matter on the usability of the system.

7. Conclusion

Information on the Internet has become richer than ever. A good human-machine interface is the key to presenting information effectively. Virtual reality is one such technology that can potentially change the way people interact with computers. In recent years, VRML has gradually established its position as a standard for distributing 3D virtual scenes on the network. However, for the results to be fruitful, we believe that it is important to add intelligent assistance to the current VRML browsers. In this paper, we describe an implemented system utilizing new features in VRML and path planning algorithms to augment the current VRML browsers with the function of generating customized guided tours. We believe that such a system can serve as a good example of incorporating machine intelligence into virtual reality for providing convenient human-machine interfaces.

Bibliography

[1] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical Potential Field Techniques for Robot Path Planning," *IEEE Transactions on Systems, Man, and Cybernetics*, 22, pp224-241, March 1992.

[2] J. Barraquand and J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *The International Journal of Robotics Research*, 10(6), pp628-649, December 1991.

[3] Rikk Carey, Chris Marrin, and Gavin Bell, editors, "*The Virtual Reality Modeling Language (VRML) Version 2.0 Specification*," International Standards Organization/International Electrotechnical Commission (ISO/IEC) draft standard 14772, August 4 1996.

[4] H. Chang and T.-Y. Li, "Assembly Maintainability Study with Motion Planning," In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan, pp1012-1019, May 1995.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "*Introduction to Algorithms*", The MIT press, pp969-974, 1994.

[6] S.-D. Jang, "*The Use and Audience Research of Interactive Multimedia Tours Guide Systems in Museums*," Masters Thesis, Institute of Communication Technology, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., July 1994.

[7] J.C. Latombe, "*Robot Motion Planning*," Kluwer Publishing, pp318-334, 1990.

[8] J.P. Laumond, "Obstacle Growing in a Non-Polygonal World," *Information Processing Letters*, 25(1), pp41-50, 1987.

[9] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," *IEEE Transaction on Computers*, 32(3), pp108-120, 1983.

[10] Chris Marrin, "*Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference*," URL: http://vrml.sgi.com/moving-worlds/spec/ExternalInterface.html, January 21, 1997.

[11] J.H. Reif, "Complexity of the Mover's Problem and Generalizations," *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pp421-427.

[12] Virtual Museum and 3D Website Design, URL: http://www.indians.org/vrml/index.html.

[13] Aquarelle Project the 3D Virtual Museum, URL: http://miles.cnuce.cnr.it/cg/aquarelleold/museum.frame.html.

[14] WorldView browser, http://www.intervista.com.