Intelligent Third-Person Control of 3D Avatar Motion

Chun-Chieh Chen and Tsai-Yen Li

Computer Science Department, National Chengchi University,
64, Sec. 2, Zhi-nan Rd., Wenshan, Taipei 116, Taiwan, R.O.C.
g9425@cs.nccu.edu.tw, li@nccu.edu.tw

Abstract. Interactive control of 3D avatar motions has important applications in computer games and animations. Most of the current games directly map the inputs of a user into canned motions of the 3D character under control. However, an ideal user interface should be more like a human moving in the real life where appropriate motions are chosen according to the environment in the front. In this paper, we have developed an intelligent control interface that helps the user to look ahead a few steps and search a motion library organized in a motion graph for the most appropriate motion for execution. The search is conducted incrementally with the best effort in each frame according to a given available time budget. We have developed a best-first strategy to maintain the search tree in a way that the number of useful nodes is kept as large as possible. Experiments on several example scenes have been conducted to illustrate how this mechanism can enhance the usability of 3D avatar control from a third-person view.

Keywords: Interactive Motion Control, Intelligent Avatar Motion, Intelligent Interface, Third-Person View.

1 Introduction

The ability to control 3D avatars interactively is crucial for many applications such as 3D computer game and virtual environment. It is especially important for the control with a third-person view (such as the one shown in Fig. 1), where the whole body motion is visible and controlled by the user. However, it has never been easy to control a 3D character with many degrees of freedom (DOF) by the use of simple low-DOF devices such as mouse and keyboard [13][1]. Most computer games today map user inputs into canned motions, which are usually triggered according to some finite-state machine. The quality of these motions is good because they were acquired from real performers by motion capture techniques. The way of selecting a good motion may be hard-coded in the program for a specific environment or selected by the user interactively. However, these mechanisms all create some burdens at the design or the run-time stage.

We think an ideal user interface for interactive control of 3D avatar motions should be more like a human moving in the real environment, where he/she always looks ahead a few step to plan for a feasible motion that can achieve his/her high-level goal while avoiding obstacles. An avatar control program with an intelligent user interface

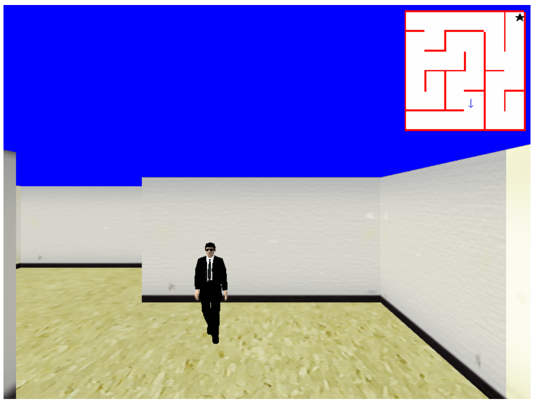


Fig. 1. Interactive avatar control from a third-person view

should release the burden of the game designer as well as the user in handling environmental constraints such as collision avoidance and automatically resolve these motion selection problems, which are less related to the game logics. In this paper, we propose an intelligent control mechanism that considers a few steps beyond the current position of the avatar and automatically selects the most appropriate motion clip from a motion library for the user. The system maintains a dynamic tree of feasible motions in real time while the user is controlling the movement. A best-first strategy has been developed to explore and maintain the most promising and useful portion of the exploration tree for a given time budget in each frame update.

We organize the rest of the paper as follows. We will review the work pertaining to our research in the next section. In Section 3, we will model the user interface problem formally. Then, we will propose our real-time search algorithm to maintain the search tree incrementally and select the best action for the next time frame. In Section 5, we will show some experimental results obtained with our system. Then the paper will be concluded with future research directions.

2 Related Work

In the literature of computer animation, animation of human figure can be generated with three categories of approaches: key-framed, dynamics, and example-based.

The key-frame based approaches use interactive authoring software or procedures, or rules to create key-frames and appropriate interpolation between key-frames [2][15]. It is tedious to generate animation of human figures with key-frames even with the sophisticated animation software available today. If the key-frames are generated in a procedural manner, then the generated animation could be made adaptive to environmental constraints such as moving on an uneven terrain.

The dynamics approaches model the dynamics properties of a human character and use simulation and control to create animations [6][14]. An advantage of this type of approaches is that the generated animations could be realistic and flexible if the physics model is correctly constructed. However, a physically sound model is not easy to build, and the performance of simulation is not real-time in general.

The example-based animation is the most popular approach in recent years. It usually refers to the animations acquired by the technique of motion capture. The advantage of this technique is that the captured motions are reproductions of real human motions and thus are the most realistic. However, the captured motions lack flexibility in situations where the human model or the environment is different from the time when the motions were captured.

Recent work in the computer animation literature strives to solve the above problems to make captured motions more reusable for different characters and environments. For example, a common approach is to use spacetime constraints or optimal control strategies to retarget the motion for the model of a new character [4][5]. Synthesizing a new sequence of motions from a motion library is also a common objective. Kim et al. used signal processing techniques to segment motions into elementary clips that can be used to synthesize new motions [8]. Gleicher organizes elementary motion clips into a motion graph that connect nodes (motion clips) with similar connection frames and use motion wrapping and blending techniques to generate smooth transition between two motion clips [9]. Lee et al. [10] organized the motion clips in a two-level graph, where the higher level organizes the type of motions into groups and the lower level selects a motion clip in the chosen group.

Due to the limited degrees of freedom available on common input devices of a desktop computer, several intelligent user interfaces have been proposed to make the task of 3D navigation or character motion control easier. For example, in [12] and [11], motion planning techniques and virtual force field have been used to improve navigation efficiency. In [7], a real-time motion planning algorithm was used to generate compliant motions for an animated character to avoid obstacles. The motions were generated at run time but limited to the upper-body motions generated procedurally. In [10], Lee et al. used three types of interfaces (menu, sketching, and images) to help a user select appropriate motion clips from a motion library. The menu interface prompts the user with a list of candidate motions at the end of a clip while our approach constantly looks ahead a few steps in every frame and selects the most appropriate motion automatically.

3 Problem Modeling

In this section, we will first review how the third-person view is used to control the animated avatar in common computer games. Then we will introduce the concept of motion graph, which is used to organize the motion library for synthesis purpose.

3.1 User Inputs

The third-person view is a common control interface for many 3D games such as Bio Hazard and Final Fantasy. For a given virtual scene, the designer usually place several cameras at appropriate locations of the scene to cover all regions that may be visited, as shown in Fig. 2. A typical game with the third-person control switches between these cameras automatically when the avatar moves. The user uses simple control

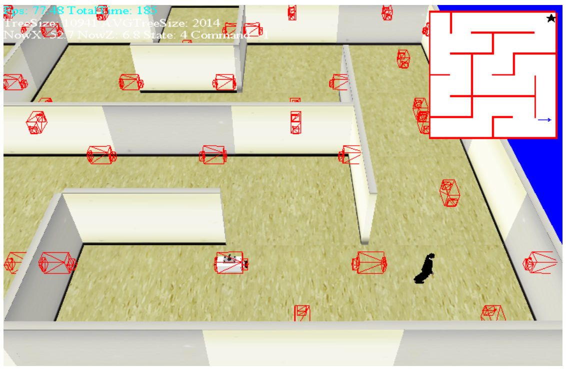


Fig. 2. Automatic transition of cameras from a third-person view

devices such as keyboard (arrow keys) or mouse (dragging) to specify the desired direction for the animated character to follow. Since the direction is specified with respect to the virtual cameras instead of the avatar, this type of control is usually called the *third-person control*. The direction for movement specified in the camera coordinate system is transformed into the avatar's local coordinate system in order to determine the next motion to take.

Most games adopt a finite state machine to map the user's inputs into the avatar's motions according to the current location and state of the avatar. If the game allows the avatar to move in an unconstrained manner, some real-time collision detection mechanism usually needs to be employed to prevent the avatar from running into obstacles. When a collision is detected, some special motion sequence is used to avoid the situation or the user needs to back up the avatar by taking some awkward maneuvers. It is awkward because in our daily life, while moving in a cluttered environment, we usually look ahead a few steps in the front to select an appropriate motion to avoid bumping into obstacles. We believe that an intelligent interface for 3D avatar control also needs to support this feature to make navigation easier.

3.2 Motion Graph

Usually after the game logic is finalized, the desired motions of the avatar are then generated by 3D animation software or motion capture devices. These generated motions are only good for this game. As mentioned in the previous section, much work in the literature of computer animation strives to develop technologies that can reuse a large library of motion data. Motion graph is one way to organize the motions in the library for motion synthesis. Each node in a motion graph represents a motion clip in the library while the edge between two nodes means that a smooth transition between the two motion clips is feasible.

A common way to make use of the motion graph is to develop an interface that allows animation designers to specify high-level constraints (such as a rough path trace) and let the system synthesize a sequence of motions from the motion library that can satisfy the given constraints. Most of the previous work makes use of motion graph in an off-line manner. In our work, we hope to develop an intelligent interactive

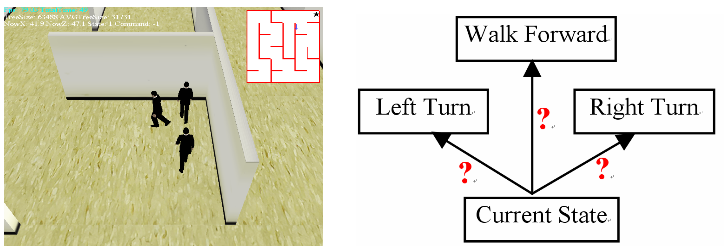


Fig. 3. Problem of selecting an appropriate motion clip with a third-person control

interface that can incorporate real-time planning into the control loop of the user interface in order to facilitate the navigation of a 3D avatar. The system should be able to look ahead a few steps and choose the most appropriate motion for the avatar according to the inputs from the user, the available motion clips in the motion library, and the obstacles in the virtual environment. For example, in the scenario of Fig. 3, the avatar is at a decision point of moving forward, turning left or turning right. For a smooth motion control, the system should be able to look ahead a few steps and select the motion clip that obeys the command of the user in principle while avoiding running into obstacles. In this case, making a left turn is the only feasible choice.

4 Feasible Motion Tree (FMT)

The system is given a motion library that is suitable for an avatar to perform the required tasks in a game. The user specifies the desired movement for the avatar from a third-person view. The main problem, as described in the previous section, is how to select a good motion clip from the library according to the user input and the environmental constraints. In the following subsections, we will first describe how we incrementally maintain a feasible motion tree in real time and then the best-first strategy that has been developed to keep the motion tree as useful as possible over time.

4.1 Maintaining Motion Tree in Real Time

As the computer hardware becomes more powerful, extra CPU cycles can be used to increase fidelity or ease of use of an application. It is generally accepted that the minimal frame rate for a 3D game is 15 frames per second (fps). If the frame rate of the game in a computer is higher than this value, the extra CPU cycles can be exploited to assist navigation. We define *feasible motion tree* as an exploration tree of the motion graph. A node, denoted as $N_k=(Mi,Pj)$, in a feasible motion tree contains the candidate motion clips and the positions (3D coordinates) of the avatar's skeleton root for executing the clips. The root of the tree is the current motion and the nodes below it are the motions that may be selected from the current motion clip in the next few steps.

For example, in Fig. 4(a), assume that we are given a motion graph consisting of six nodes. A directed arc between two nodes represents a feasible transition from the source to the sink. For example, in Fig. 4(b), M2, M3, and M4 are the possible next

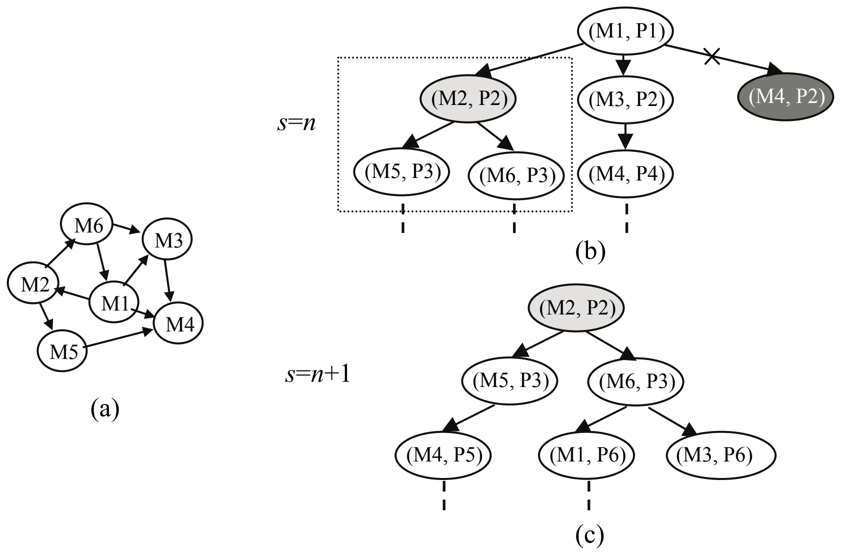


Fig. 4. For the given motion graph in (a), maintaining feasible motion tree at steps (b) *s*=*n* and (c) *s*=*n*+1

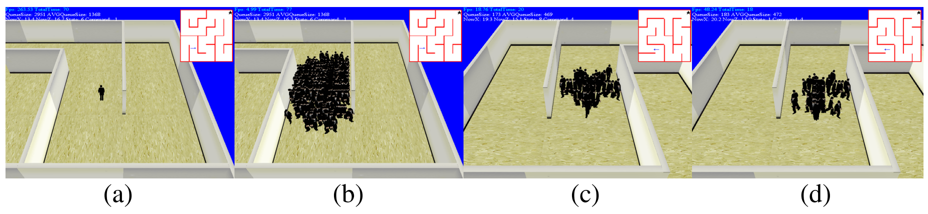


Fig. 5. Feasible motion trees maintained at different stages (a) initial state (b) after several frames of idle (c) starting moving (d) maintaining tree while moving

motion clips according to the motion graph in Fig. 4(a). However, assume that executing M4 from location P2 results in a collision and makes (M4, P2) infeasible. M2 and M3 are two feasible motions that can be used for the next step. When the time is allowed in every frame, the tree will continue to grow as more motions are tested for feasibility for the subsequent steps. As the motion clip for the current step is finished, the most appropriate next motion clip in the feasible motion tree is selected according to some criteria described in the next subsection. When the next motion clip starts, the root node of the tree moves downward and the sibling subtrees are pruned accordingly, as shown in Fig. 4(c).

An example of feasible motion tree in our experiments is shown in Fig. 5. The avatar started at a position shown in Fig. 5(a) where the feasible motion tree contains only the root node. The tree grew incrementally by using the spare time of each frame after finishing graphical rendering at a user-specified minimal frame rate. Since the user did not issue commands to move the avatar for a while, the tree grew to a significant size after a couple of frames as shown in Fig. 5(b). Sampled frames in the motion

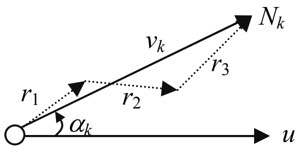


Fig. 6. The command matching method for computing the priority of a node N_k

clips maintained in the tree were overlaid on the screen to show the size of the tree. As the avatar started to move in Fig. 5(c), the tree was pruned accordingly as the root of the tree moved down. Once a new motion clip was chosen for execution in the next step, only the selected branch of the tree remained and new leaves continued to grow on this branch, as shown in Fig. 5(d).

4.2 Exploration Strategies

Given a limited amount of time for exploration in each frame, the quality of the selected motion depends on the strategy of exploration in the motion graph. A straightforward strategy is a breadth-first approach that explores the motion graph in a breadth-first fashion. However, we think a good exploration strategy should be able to grow the most useful branch of the feasible motion tree. For example, the criteria for selecting the best node for growing should consider the depth of the node in the tree as well as how the motion fits the current input of the user. There are many possible ways to design the criteria of goodness. In this paper, we have designed a best-first strategy as described below.

We assume that each node is associated a base priority p_b, where $0<p_b<1$. This priority reflects the likelihood of this node being used in the future movement of the avatar according to the input command of the user. We first transform the input command, specified as a vector, u, in the camera space into the coordinate system of the avatar. For each motion clip, we compute the vector of relative movement, r, by taking the difference of the avatar positions before and after executing the motion clip. As illustrated in Fig. 6, for a given node N_k, in the feasible motion tree, the movement (v_k) relative to the current position can be computed by taking the sum of r of the nodes along the path, denoted by σ_k, backtracked to the root node. That is, $v_k = \sum_{\sigma_k} r_i$.

The angle α_k between u and v_k indicates how the resulting motion matches the current input of the user. We then use a function g to map α_k to p_{bk} in [0, 1]. That is, $p_{bk} = g(\angle(u, v_k))$. We call this way of computing the priority the *command matching method*. In the spare time of each frame, we select the leaf node with the highest priority for exploration. The priority, p_k, of a leaf node, N_k, is computed by multiplying p_{bk} of all nodes along the path σ_k to the root (That is, $p_k = \prod_{Ni \in \sigma_k} p_{bi}$). Therefore, the lower the nodes in the tree or the less they matched the input command, the lower priority a leaf node will get.

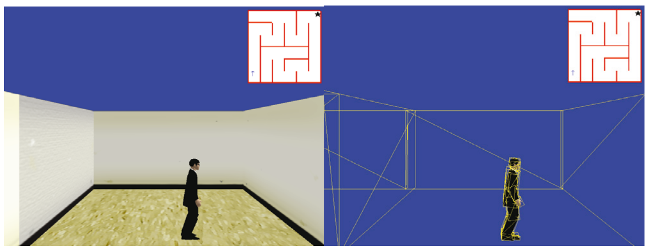


Fig. 7. Bounding box model used for collision detection

4.3 Selection Criteria for the Next Motion

When the current motion clip comes to the end, the system selects a most appropriate clip for the next step. The criteria for selecting the next motion clip in our system is based on the information collected in the current feasible motion tree as well as the input command from the user. We assume that a good choice of motion for the next step should be able to guarantee that the next few steps ahead are feasible according to the current input. Since the priority value of each node in the tree reflects how the resulting motion matches the user command and we have grown the tree according to the feasibility and priority of the leaf nodes, the structure of the feasible motion tree is a good indicator for selection. We compute the weighted sum of the priorities values for all the nodes in a branch and select the child with the highest sum among all children of the root. The weights of the priorities are smaller at lower levels of the tree, and their values are determined empirically for now.

5 Experiments

5.1 Implementation and Experimental Settings

We have implemented the intelligent interface for avatar control in a 3D navigation program based on the Direct3D library on the Windows platform. Keyboard is the input device used in the experiments. The motion clips used in our experiments are modified from the motions in the motion capture database provided by the CMU graphics laboratory [3]. The collisions between the avatar and the environment are detected by using the method based on the axis-aligned bounding boxes (AABB) [16] of the geometric models of the objects in the scene, as shown in Fig. 7. The avatar is checked for collisions in every frame of a motion clip, and a motion clip is valid only if all of the frames in it are collision-free. The 3D scenes used in our current experiments are maze-like virtual environments such as the one shown in Fig. 8.

We have conducted extensive experiments to study the effects of the proposed intelligent control interface. However, it is difficult to evaluate the system through repetitive experiments since the commands used by a user to move from one point to another may be different at different times. The learning effects are also difficult to avoid as the user gets familiar with the environment through practice. According to the experiments of using an intelligent user interface from the first-person control

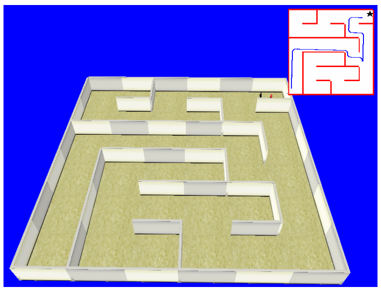


Fig. 8. An example scene used in the experiments

reported in [12], we assume that adopting a look-ahead mechanism with the spare CPU cycles in a frame update is helpful for navigation control from a third-person view. Therefore, instead of measuring the efficiency of navigation (total time to accomplish a navigation task) or the subjective experience of the user, we measure the average size of the feasible motion tree along a recorded path for different exploration strategies. The problem of effectiveness becomes how good the look-ahead information can be maintained to support a good navigation decision.

In the experiments, we first asked the user to navigate through the scene from an initial position to a designated goal position. The user commands and the trajectory were recorded for replay in future experiments. An example trajectory is shown at the upper-right corner of Fig. 8. In each of the subsequent experiments, we use the same input commands to reproduce the same navigation path. The number of nodes in the feasible motion tree in each frame is recorded in each experiment, and the average tree size is computed at the end. Although the input commands and navigation path are the same, the average size of the feasible motion tree is different because of different exploration strategies.

5.2 Experimental Results

In our experiments, we have tested five different strategies for exploration of the motion graph, and the result is shown in Fig. 9. The horizontal axis is the interval between two frames, which is the inverse of frame rate. The larger the interval, the more the spare CPU cycles are used to grow the feasible motion tree. The vertical axis is the average number of nodes that are maintained in the tree during the course of the navigation. The first exploration strategy (Type 0) is the breadth-first strategy. The second and the third strategies (Type 1 and 2) use the command matching method with and without re-sorting the priorities of the leaf nodes in each frame, respectively. Type 3 uses user-defined priorities based on the statistic data of how the children of a node are chosen in a navigation task conducted in the given environment. Type 4 uses a hybrid approach by combining the priorities of Type 1 and Type 3.

The results in Fig. 9 reveal that the size of the tree grows linearly with the available time in each frame update. This means that on slower machines, one can expect that the size of the tree will be smaller and the effect of looking ahead for motion selection can degrade gracefully. For these five types, Type 0 has the smallest average tree size

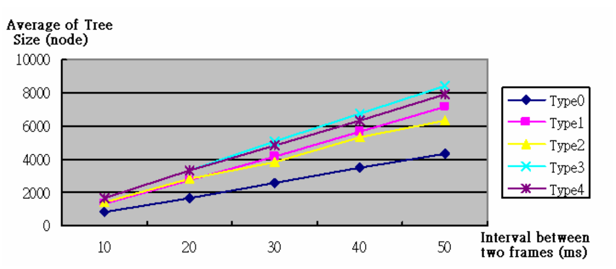


Fig. 9. Evaluation of different exploration strategies

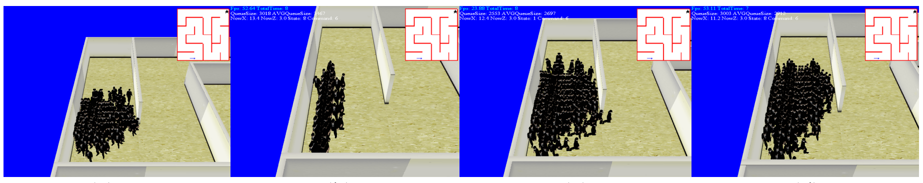
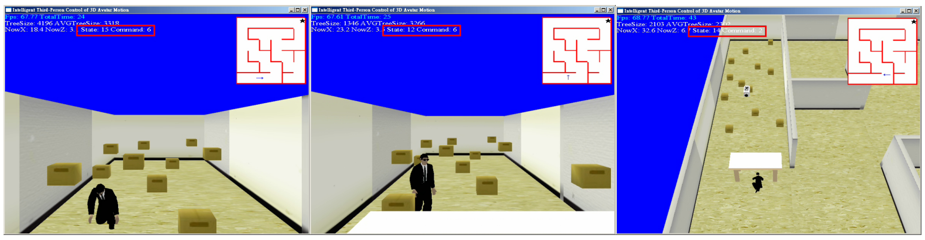


Fig. 10. Visited avatar positions in the feasible motion trees with different exploration strategies: (a) breadth-first (b) command-matching (c) user-defined (statistic) (d) hybrid

as expected because of its breadth-first strategy. Type 1 and Type 2 (command matching) all perform better than Type 0, and re-sorting the priority list in every frame is important for maintaining a good shape of the tree since the user inputs may change from time to time. Type 3, the user-defined method, has the best performance since the priorities were computed according to statistic data from the history. However, the setting is specific to the given scene and may not work well for other scenes. Type 4 is the hybrid approach combining Type 1 and Type 3 and the effect of this type also falls between these two types as expected.

In Fig. 10, for a given current position of the avatar, we show all the future positions that have been maintained in the feasible motion tree with various strategies. In Fig. 10(a), the shape of the visited region is more evenly distributed because the breadth-first approach is used. In Fig. 10(b), the adopted command-matching method prefers growing in a specific direction because of the consistent commands of walking forward in the past few steps. In Fig. 10(c), the user-defined priorities are used and the size of the tree is the largest. Finally, the tree of using the hybrid approach combining 10(b) and 10(c) is depicted in Fig. 10(d).

In Fig. 11, we show another example scene where obstacles were placed at various locations of the maze to make the movement of the avatar more difficult. In order to facilitate the movement of the avatar through the space, we enrich the motion graph by adding a “crouch-moving” motion and its transition motions to other motions. In Fig. 11(a), the system had chosen a crouch-moving motion for the avatar to pass the obstructive obstacle. In Fig. 11(b), the obstructive obstacle was lower and the avatar chose to make a turn to pass through the region. Finally, the avatar used the crouch-moving motion again to pass underneath the table placed in his way, as shown in Fig. 11(c).


Intelligent Third-Person Control of 3D Avatar Motion
Intelligent Third-Person Control of 3D Avatar Motion
Intelligent Third-Person Control of 3D Avatar Motion

```
Fps: 67.77 TotalTime: 24
TreeSize: 4196 AVGTreeSize: 3318
NowX: 18.4 NowZ: 3.  State: 15 Command: 6
```

```
Fps: 67.61 TotalTime: 25
TreeSize: 1346 AVGTreeSize: 3266
NowX: 23.2 NowZ: 3.  State: 12 Command: 6
```

```
Fps: 68.77 TotalTime: 43
TreeSize: 2103 AVGTreeSize: 23
NowX: 32.6 NowZ: 6.  State: 14 Command: 2
```

(a) (b) (c)

Fig. 11. Examples of versatile motions generated for the avatar to overcome environmental constraints (a) crouch moving (b) left turning (c) crouching underneath the table

6 Conclusions

It has been a long-time research goal to allow a 3D avatar to be commanded interactively with high-level inputs under the constraints of the environment. In this paper, we have developed an intelligent 3D control interface that looks ahead a few steps in order to make a better selection of the next motion clip from the given motion library. The system exploits the spare CPU cycles between two frames to maintain the feasible motion tree incrementally. We also have developed various strategies and conducted experiments to demonstrate the effectiveness of the strategies and compared the tradeoffs of these strategies.

In the current work, we assume that the interface with the look-ahead mechanism should work better than the one without this feature since similar claims have been made before. In the future, we would like to conduct further experiments to measure the improvement of the navigation efficiency more precisely. In addition, we will conduct more experiments to study how to select a good cut-off depth for tree growing such that a good balance between frame rate and tree size can be obtained.

Acknowledgments

This research was funded in part by the National Science Council (NSC) of Taiwan under contract no. NSC 95-2221-E-004-015.

References

1. Bowman, D., Kruijff, E., LaViola, J., Poupyrev, I.: 3D User Interfaces: Theory and Practice. Addison-Wesley, Boston (2004)
2. Bruderlin, A., Calvert, T.W.: Goal-Directed Dynamic Animation of Human Walking. In: Computer Graphics (Proc. of SIGGRAPH 89) vol. 23, pp. 233–242 (1989)
3. CMU Graphics Lab Motion Capture Database. http://mocap.cs.cmu.edu/
4. Gleicher, M.: Motion Editing with Spacetime Constraints. In: Proc. of the 1997 Symposium on Interactive 3D Graphics (1997)
5. Gleicher, M.: Retargeting Motion to New Characters. In: Computer Graphics (SIGGRAPH 98 Proceedings) pp. 33–42 (July 1998)

6. Hodgins, J.K., Wooten, W.L., Brogan, D.C., O'Brien, J.F.: Animating Human Athletics. In: Proc. of SIGGRAPH 95, pp. 71–78 (1995)
7. Hsu, H.W., Li, T.Y.: Third-Person Interactive Control of Humanoid with Real-Time Motion Planning Algorithm. In: Proc. of IEEE International Conf. on Intelligent Robots and Systems, IEEE Computer Society Press, Los Alamitos (2006)
8. Kim, T.H., Park, S.I., Shin, S.Y.: Rhythmic-Motion Synthesis Based on Motion-Beat Analysis. ACM Transactions on Graphics 22(3), 392–401 (2003)
9. Kover, L., Gleicher, M., Pighin, F.: Motion Graphs. In: Proc. of SIGGRAPH 2002 (2002)
10. Lee, J., Chai, J., Reitsma, P., Hodgins, J.K., Pollard, N.: Interactive Control of Avatars Animated with Human Motion Data. In: Proc. of SIGGRAPH 2002 (2002)
11. Li, T.Y., Hsu, S.W.: An Intelligent 3D User Interface Adapting to User Control Behaviors. In: Proc. of International Conf. on Intelligent User Interfaces (IUI'04) (2004)
12. Li, T.Y., Ting, H.K.: An Intelligent User Interface with Motion Planning for 3D Navigation. In: Proc. of the IEEE Virtual Reality 2000 Conf, pp. 177–184. IEEE Computer Society Press, Los Alamitos (2000)
13. Nielson, G.M., Olsen, D.R.: Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices. In: Proc. of the 1986 Workshop on Interactive 3D Graphics, pp. 175–182 (1987)
14. Paloutsos, P., van de Panne, M., Terzopoulos, D.: Composable Controllers for Physics-Based Character Animation. In: Proc. of SIGGPRAH2001, pp. 251–260 (2001)
15. Perlin, K., Goldberg, A.: Improv: A System for Scripting Interactive Actors in Virtual Worlds. In: Proc. of SIGGRAPH 96, pp. 205–216 (1996)
16. van den Bergen, G.: Efficient Collision Detection of Complex Deformable Models Using AABB Trees. Journal of Graphics Tools 2(4), 1–14 (1997)