Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans

James J. Kuffner, Jr Jean-Claude Latombe
Computer Science Robotics Lab
Stanford University
Stanford, CA 94305-9010, USA
{kuffner, latombe}@cs.stanford.edu

Abstract

This paper presents a simple and efficient method of modeling synthetic vision, memory, and learning for autonomous animated characters in real-time virtual environments. The model is efficient in terms of both storage requirements and update times, and can be flexibly combined with a variety of higher-level reasoning modules or complex memory rules. The design is inspired by research in motion planning, control, and sensing for autonomous mobile robots. We apply this framework to the problem of quickly synthesizing from navigation goals the collision-free motions for animated human figures in changing virtual environments. We combine a low-level path planner, a path-following controller, and cyclic motion capture data to generate the underlying animation. Graphics rendering hardware is used to simulate the visual perception of a character, providing a feedback loop to the overall navigation strategy. The synthetic vision and memory update rules can handle dynamic environments where objects appear, disappear, or move around unpredictably. The resulting model is suitable for a variety of real-time applications involving autonomous animated characters.

1 Introduction

Graphic Animation by computer lies at the boundary of modeling and simulating the real world, and shares much in common with the design and control of robotic systems. This paper approaches the problem of generating motion for animated characters from a robotics perspective. More specifically, the problem of controlling an autonomous animated character in a virtual environment is viewed as that of controlling an autonomous virtual robot complete with virtual sensors.

This paper combines the fast 2D path planner and controller originally presented in[13], with a simple synthetic vision and memory model to compute natural-looking motions for navigation tasks. The controller is used to synthesize cyclic motion capture data for an animated character as it follows a computed path towards a goal location. The goal location can be can be user-specified or defined by a behavior script. The rendering hardware is used to simulate the visual perception of a character so that it reacts in response to obstacles in its visual field. A record of perceived objects and their state is kept as the character explores an unknown virtual environment, allowing the character to construct motion plans based solely on its own internal model of the virtual world. The internal model is updated as new sensory information arrives, and a new navigation plan is computed if necessary. The resulting animation can be generated at interactive rates and looks fairly realistic.

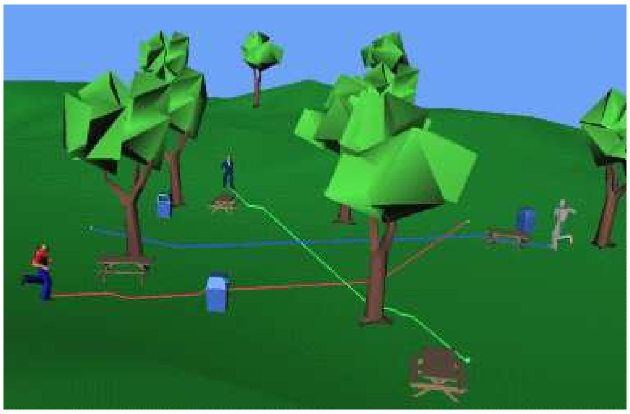

Figure 1. An interactive session involving multiple characters navigating outdoors.

2 Related Work

Previous work in motion synthesis for animated characters has traditionally been divided between real-time applications and off-line animation production. However, as processor speeds continue to increase, algorithms originally intended for off-line animations will gradually become feasible in real-time virtual environments.

Much research effort in robotics has been focused on designing control architectures for autonomous agents that operate in the real world[15, 7, 1]. More recently, many of these same techniques have gradually been adapted for the purposes of creating autonomous animated characters in virtual worlds. The ultimate goal being the creation of fully-autonomous, interactive, artificial agents. Reactive behaviors applied to simple simulated creatures appeared in the graphics literature with Reynolds' BOIDS model[22]. Tu and Terzopoulos implemented a realistic simulation of autonomous artificial fishes, complete with integrated simple behaviors, physically-based motion generation, and simulated perception[29]. Noser, *et al.* proposed a navigation system for animated characters using ideas for synthetic vision originally developed by Renault, *et al.*[21]. Their implementation also included memory and learning models based on dynamic octrees[17]. These ideas were later expanded to include virtual tactility and audition[18, 27]. Researchers at the University of Pennsylvania have been exploring algorithms for controlling their Jack human character model[10, 8], incorporating body dynamics[12], and high-level scripting[2]. Koga *et al.* combined robot motion planning and human arm inverse kinematics algorithms for automatically generating animation for human arm manipulation tasks[11]. Other systems include Perlin and Goldberg's Improv software for interactive agents[20, 19], the ALIVE project at MIT[5, 16], Johnson's WavesWorld, the Oz project at CMU[3], and the work of Strassman[24, 25], and Ridsdale, *et al.*[23]. Researchers at Georgia Tech have combined physically-based simulation with group behaviors for simulating human athletics[6]. They have also designed a controller for human running in 3D[9]. Despite these achievements, building autonomous agents that respond intelligently to task-level commands remains an elusive goal, particularly in real-time applications.

Previous researchers have argued the case for employing some kind of virtual perception for animated characters[21, 28]. The key idea is to somehow realistically model the information flow from the environment to the character. Giving each character complete access to all objects in the environment is both conceptually unrealistic, and can be impractical to implement for large environments with many objects. One way to limit the information a character has access to, is to consider only objects within a sphere centered around the character[22]. However, most characters of interest (including human characters) do not have such omnidirectional perception. Rather, sensory information from the environment flows from a primary direction, such as the cone of vision for a human character. Synthetic audition may often be considered to be omnidirectional, but for tasks involving navigation and obstacle avoidance, some kind of synthetic vision is needed.

3 Synthetic Vision

There have been several previous proposals for modeling simulated visual perception. Tu and Terzopoulos implemented a synthetic vision for their artificial fishes based on ray-casting[29, 28]. Blumberg experimented with image-based motion energy techniques for obstacle avoidance for his autonomous virtual dog[4]. Terzopoulos and Rabie proposed using a database of pre-rendered models of objects along with an iterative pattern-matching scheme based on color histograms for object recognition[26]. Noser, *et al.* presented a clever synthetic vision model that uses object false-coloring and dynamic octrees to represent the visual memory of the character[17].

We are primarily interested in synthetic vision techniques that are practical for real-time systems. Specifically, our goal is to allow an autonomous character endowed with synthetic vision to explore an unknown interactive environment, while maintaining a visual memory or "cognitive map" of what it has perceived. This map may then be used as input to a planning or navigation algorithm. Due to the time constraints inherent in real-time systems, the synthetic vision module must be reasonably fast, and the visual memory model must be simple and efficient to update. In addition, we require the vision and memory modules to handle changing environments, where objects can appear, move, or disappear without warning.

The function of the synthetic vision module is to determine the set of objects currently visible to a character, given the environment scene description along with a specification of the character's current viewing frustum. Exact geometric algorithms for 3D visibility are complex, and suffer from poor performance relative to hardware Z-buffers. This is especially true for large scenes with moving obstacles, which typically cannot be pre-processed. We adopt an approach to synthetic vision similar to the one described by Noser, *et al.*[17, 18]. The general idea is to render an unlit model of the scene (flat shading) from the character's point of view, using a unique color assigned to each object or object part. The pixel color information is extracted to obtain a list of the currently visible objects. As pointed out by Thalmann, *et al.* in [27], synthetic vision differs from vision computations for real robots, since we can skip all of the problems of distance detection, pattern recognition, and noisy images. This allows us to implement a reasonable model of visual information flow that operates fast enough for real-time systems. Furthermore, since the object visibility calculation is fundamentally a rendering operation, all of the techniques that have been developed to speed up the rendering of large, complex scenes can be exploited. This includes scene-graph management and caching, hierarchical level-of-detail (LOD) approximations, and frame-to-frame coherency.

From the list of currently visible objects computed by the vision module, a set of observations is formed by combining this list with each object's current location. Finally, this set of observations is added to the character's internal memory model of the environment, and a navigation plan is computed. The navigation planning algorithm we use combines a fast path planner, motion controller, and cyclic motion capture data, and is presented in detail in[13]. However, any appropriate real-time navigation planning strategy may be used.

For our system to work, we assume the environment is broken up into a collection of small to medium-sized objects, each assigned a unique ID. For example, a single object may be a chair or a door. Large objects such as walls or floors are further subdivided, and each piece is assigned an ID. This assignment may be done automatically using any reasonable object-level spatial subdivision technique. A color table is initialized to represent a one-to-one mapping between object IDs and colors. To check which objects are visible to a particular character, the scene is rendered offscreen from the character's point of view, using flat shading and using the unique color for each object[1] as defined by the object ID (see Figure 3). The size of the rendered image need not be very large (usually 200x200 pixels yields sufficient detail). The resulting image pixels are scanned, and a list of visible objects is obtained from the pixel color information. This list may then be combined with other environment state information to encode higher-level aspects of a character's perception. Possible examples include encoding semantic information about certain objects, velocities of objects, and relationships between objects.

4 Internal Representation and Memory

Each character maintains an internal model of the world as it explores a virtual environment. Noser, *et al.* used an occupancy grid model (e.g. an octree) to represent the visual memory of each character[17]. We instead rely upon the object geometry stored in the environment along with a list of object IDs and their most-recently observed states.[2] This provides a compact and fast representation of each character's internal world model that is scalable to large environments with many characters.

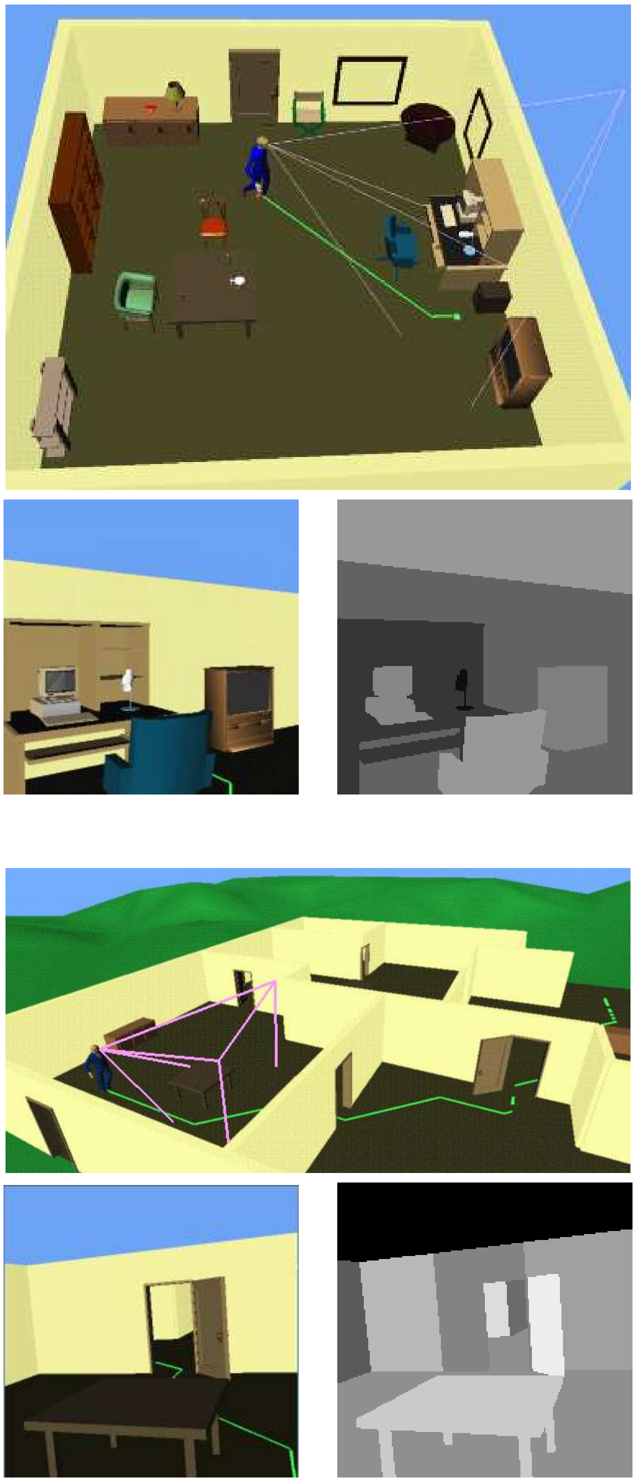

Figure 2. A character navigating in a virtual office. The image pairs below each scene view depict renderings from the character's point of view using both the true-color, lighted models, and the unlit, false-color visibility models.

[1] Note that this color is used only when rendering the visibility image offscreen, and does not affect renderings of the object seen by the user, which may be multi-colored and fully-textured.

[2] Although we refer to lists in our description, we actually use arrays indexed by the object ID for faster performance. This is practical for environments of roughly 20,000 objects or less, depending upon available memory.

When a character observes the environment, the vision module returns the set of object IDs representing objects that are currently visible. Each object ID is combined with other state information, such as the corresponding object's current 3D transformation. The state information of each of the visible objects is obtained directly from the environment, and the character updates its internal model of the world with each object's recently observed state. Previously unobserved objects are added to the character's list of known objects, and the rest of the visible objects are simply updated with their current state. Objects that are not currently visible but have been observed in the past, retain their most recent previously-observed states. This process maintains a kind of spatial memory for each character.

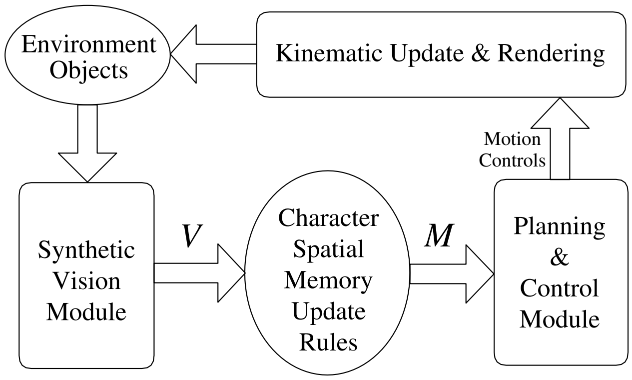

Figure 3. The basic sense-plan-control loop for static environments.

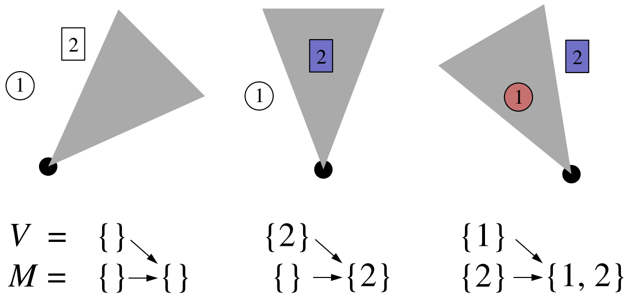

Figure 4. Updating $\mathcal{M}$ in static environments

5 Perception-Based Navigation

Using such a sensing and memory model, a character can be made to explore in real-time an unknown environment, and incrementally build its own internal model of the world. We now describe the algorithm in more detail. First, we will describe the basic sense-plan-control loop that works for static environments. We then show how to modify the basic algorithm to work for dynamic environments.

Let $\mathcal{O}$ denote the set of all objects in the environment. Each character maintains a set $\mathcal{M}$ of *observations* built incrementally from the output of the vision module. Observations are represented as tuples $\langle objID_i, \mathcal{P}_i, T_i, v_i, t\rangle$, with components:

$objID_i$	the object ID of object i
$\mathcal{P}_i$	properties of object i
T_i	3D transformation of object i
v_i	velocities of object i
t	observation time

The set $\mathcal{P}_i$ contains properties of the object, including semantic information about the object, or characteristics pertinent to its current state. For example, if the observed object is a door, then $\mathcal{P}_i$ might identify the object as something that can be moved (rotated), is currently closed, is currently unlocked, etc.[3] Object properties are flexible and as explained further in Section 6, can be utilized by higher-level reasoning engines to enable the character to make more informed decisions about the world. The transformation T_i is the observed position and orientation of the object represented by $objID_i$. The component v_i contains the observed linear and angular velocities of the object. The last component t is the *time stamp*, or the time that the observation was made.

$\mathcal{M}$ represents the character's visual memory of $\mathcal{O}$. Initially $\mathcal{M}$ is empty. At regular intervals, the character's visual perception is simulated by the vision module, which renders the color-coded instances of the objects in $\mathcal{O}$. After scanning the pixels of the resulting image, the set $\mathcal{V}$ is returned, containing the object IDs of all currently visible objects. Each $objID_i$ in $\mathcal{V}$ is combined with its corresponding object's state information to form the tuple $\Omega = \langle objID_i, \mathcal{P}_i, T_i, v_i, t\rangle$. If no existing tuple in $\mathcal{M}$ contains $objID_i$, then the object was previously unknown, and Ω is added to $\mathcal{M}$. Otherwise, if there exists a tuple in $\mathcal{M}$ that contains $objID_i$, then this object was previously observed. Hence, its corresponding state information is updated based on the values contained in Ω. After $\mathcal{M}$ has been updated from $\mathcal{V}$, then the navigation path-planning module is invoked using only the objects and transformations in $\mathcal{M}$ as obstacles. Thus, each character plans a path based solely on its own learned model of the world. As the character follows the path, new objects are observed, and $\mathcal{M}$ is updated by repeating the above process, and a new path is computed. The data flow of the algorithm is shown in Figure 3. Figure 4 illustrates how $\mathcal{M}$ is incrementally updated as previously unknown objects are observed.

The aforementioned procedure will work just fine for en-

[3] For storage efficiency, permanent (unchanging) object properties can be stored in a global table indexed by objectID, while variable properties can be encoded with observation bit flags.

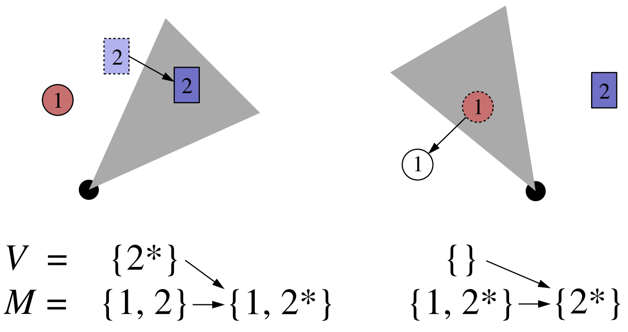

Figure 5. Updating $\mathcal{M}$ in dynamic environments

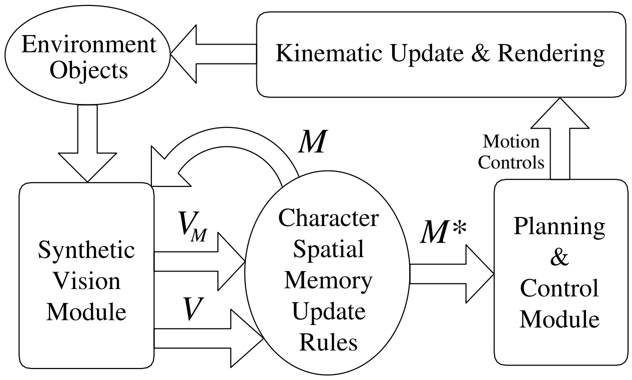

Figure 6. The revised sense-plan-control loop for dynamic environments.

vironments with static objects, but we need to make a minor modification to correctly handle changing environments where objects can appear, disappear, or move around unpredictably. The problem lies in recognizing when a previously observed object has disappeared (or moved) from its previously observed location. As an example, let us consider the following two scenarios:

- A character observes an object o_2 at a location T_2. Later, o_2 moves to a new location T_2', and is again observed. In this case, the state information for o_2 in $\mathcal{M}$ should be simply updated to reflect the new location.
- A character observes an object o_1 at a location T_1. Later, o_1 moves to a new location T_1'. However, before the new location is observed, the character observes its former location (o_1 is now missing). In this case, the previous observation of o_1 in $\mathcal{M}$ should be deleted or invalidated.

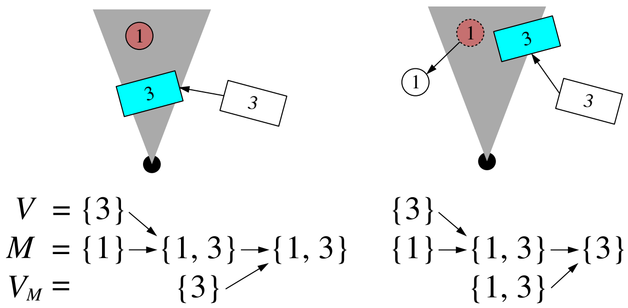

Figure 7. "Obscured" vs. "Missing" scenarios requiring $\mathcal{V}_M$ in order to distinguish between them and update $\mathcal{M}$ properly.

These two scenarios are depicted in Figure 5. The left side illustrates the case where a previously observed object(2) moves to a new location, and the new location is observed before the former location. The state information in $\mathcal{M}$ for the object is updated to account for the new observation(2*). The right side of Figure 5 depicts the case where a previously observed object(1) moves to a new location, and its former location is observed prior to its new location. The character should realize that the object is now missing and remove or invalidate the observation in $\mathcal{M}$.

How can the vision module determine whether an object is truly missing, rather than simply obscured by another object? The solution is to re-run the vision module after $\mathcal{M}$ has been updated using only the objects contained in $\mathcal{M}$, along with their corresponding transformations. The result is a set $\mathcal{V}_M$ of object IDs that corresponds to the set of objects the character *expects* to see based on $\mathcal{M}$. By comparing $\mathcal{V}$ and $\mathcal{V}_M$, we can distinguish the above case. Specifically, let $\mathcal{X} = \mathcal{V}_M - \mathcal{V}$ be the set of all object IDs contained in $\mathcal{V}_M$ but not in $\mathcal{V}$. Thus, $\mathcal{X}$ corresponds to objects that the character concludes have disappeared or moved from their previously observed locations, but their new locations are unknown. Consequently, all observation tuples in $\mathcal{M}$ containing object IDs in $\mathcal{X}$ must be removed or invalidated. Incorporating this change into the algorithm facilitates its use in arbitrarily changing environments. The data flow diagram of the modified algorithm is shown in Figure 6. Two scenarios where the lists of visible objects($\mathcal{V}$) are the same, but the final updates to $\mathcal{M}$ are different, are depicted in Figure 7. The scenario on the left corresponds to the *obscured* case, where the previously-observed object(1) is hidden by the appearance of a new object(3). $\mathcal{V}$ and $\mathcal{V}_M$ are equal in this case, so $\mathcal{M}$ retains its previous observation of object(1). The scenario on the right corresponds to the *missing* case, where the previously-observed object(1) has moved to an unknown location while a new object(3) has appeared. Since $\mathcal{V}$ and $\mathcal{V}_M$ differ, the previous observation of object(1) is removed from $\mathcal{M}$.

6 Learning and Forgetting

The memory update rules described in Section 5 represents a very simple model that remembers all observations until sensing contradicts them. We refer to this as the *basic model*. However, the framework proposed allows several possible memory models to be used. For example, one can imagine a *temporal model* that remembers observations for a certain period of time. In this case, old observations are periodically deleted from the character's memory. Alternatively, deleting old observations may depend upon the properties of the object. For example, if a non-movable object such as a wall or a floor were observed, we may wish to always retain that observation, while if a moving object such as a vehicle or animal were observed, we may decide to expire such observations as time passes. This means that there can be different types of objects in the world, and different memory rules for each type. We refer to this type of model as a *logical* or *deductive model*, since updating the character's memory involves logical rules regarding observations of certain objects or object types.

Above low-level navigation planning, there is a layer of reasoning that makes decisions about the goals and beliefs of the character based on its internal model of the world. There are many proposed architectures for logic and reasoning in the artificial intelligence literature, each with advantages and disadvantages. For the purpose of creating autonomous characters, any suitable AI architecture will suffice given that is operates efficiently and provides the ability to encode memory rules. The rules can then be made arbitrarily complex. For example, suppose that a character is exploring an unknown maze of connecting rooms with the goal of finding an exit. After some time, suppose the character concludes that no exit exists based on what it has previously observed (i.e. the low-level navigation planner fails). This event may then trigger some higher-level reasoning that will allow the character to continue to explore (e.g. there is no path, but some doors which have been seen closed or locked recently may have been opened in the meantime. Therefore, observations about closed doors may be deleted, despite the fact that they are not yet old enough to discard/forget, etc).

7 Experimental Results

The algorithms described in this paper have been implemented and tested on an SGI InfiniteReality2 running Irix 6.2. Interactive performance has been achieved, on complex scenes with up to three characters. During a session, the user can click and drag on goal locations or obstacles, and the low-level path planner will calculate an updated, minimal-length, collision-free path (if one exists) in approximately one-tenth of one second on average. The path is

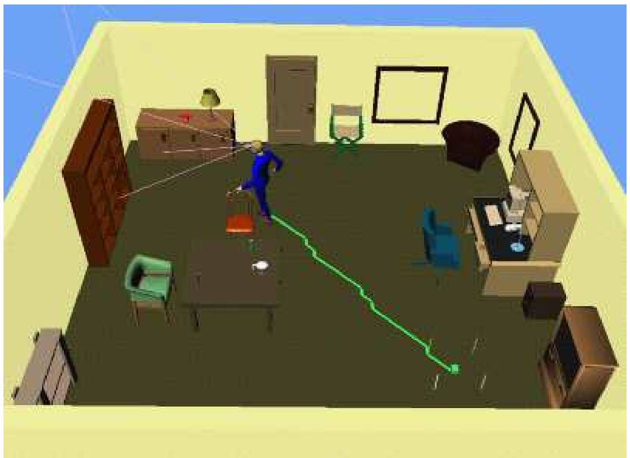

Figure 8.

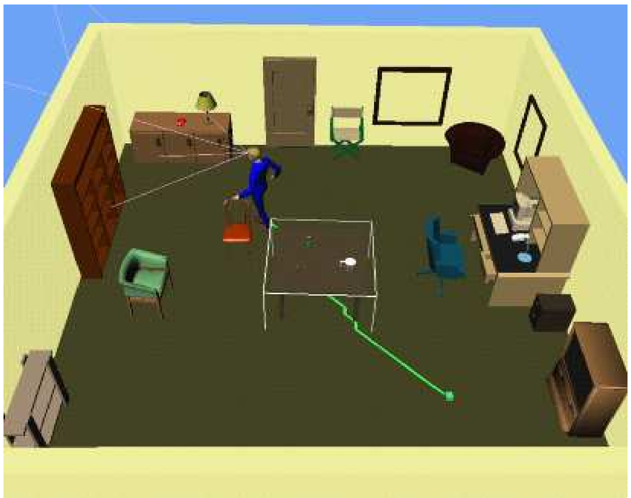

Figure 9.

then sent directly to the controller, and the character will immediately begin following the new path. Since the goal is allowed to move arbitrarily, it is possible for one character to perform simple tracking or following of another character. This can be useful for pursuit games, or having a group of characters follow a tour guide.

For the purposes of path following, the simple PD control algorithm described in [13] was implemented for a human-like character with 17 joints. Two sets of motion capture data were used in the experiments: a walk cycle and a jog cycle. Sample output involving an outdoor environment is illustrated in Figure 1. Multiple characters were run simultaneously, each planning around the other characters as they followed their own computed paths and built-up internal memory models of the environment. Figure 8

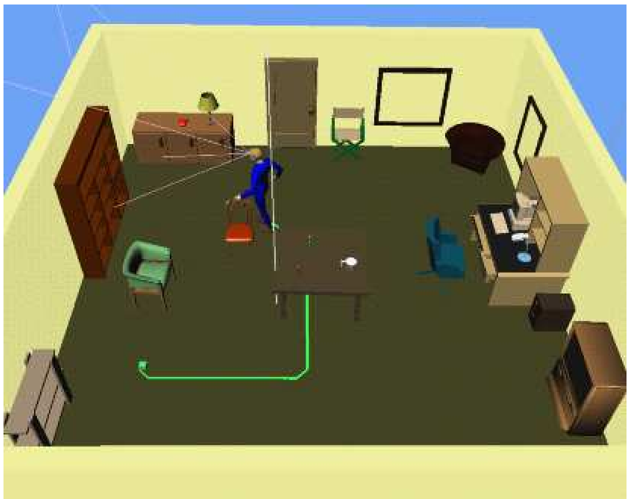

Figure 10.

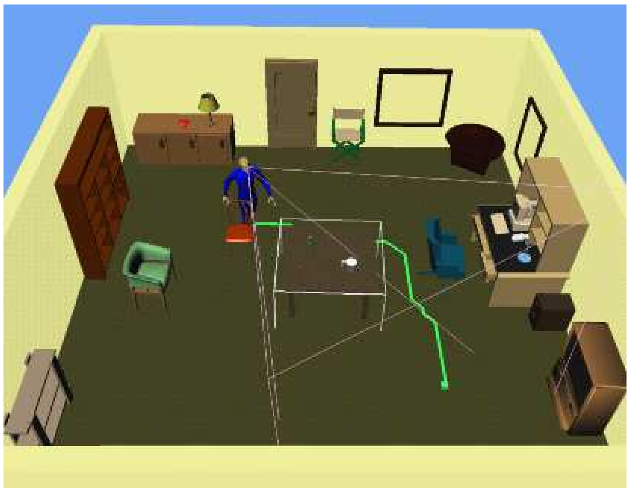

Figure 11.

shows a human character after building a complete cognitive map of a room. Figure 9 shows a previously-observed table being moved to a new location behind the character's back. Since the character is unaware of the change, its navigation path to the goal location is unaffected. Figure 10 illustrates that fact that when the goal location is moved, the character plans a path assuming the table is still at its former location. Finally, Figure 11 depicts how when the character turns around to follow the planned path, it notices the table in its new location, and plans accordingly. Figure 12 shows snapshots at various stages of an animation involving a single character exploring an unknown maze environment. The final path is solely the result of the interaction between path planning based in the character's internal model and the visual feedback obtained during exploration. Figure 13 shows a trace of the actual motion of the character during exploration. Figure 14 shows a trace of the actual motion during exploration of another unknown maze with dead ends that forced the character to backtrack.

8 Conclusion and Discussion

This paper presents a simple model of visual perception, memory, and learning for autonomous characters that is suitable for real-time interactive applications. The model is efficient in both storage requirements and update times, and can be flexibly combined with a variety of higher-level reasoning modules or complex memory rules. Research on techniques to control groups of multiple actors using the existing planning and perception modules would be useful, and is currently underway[14]. This includes some simple prediction based on observed velocities to take into account the estimated motion of other characters and obstacles during planning.

Although the synthetic vision module runs fast enough to support a small number of characters simultaneously, it is currently the bottleneck in the computation. This is primarily due to the fact that each character must render the scene *twice* in order to correctly update its cognitive map in the case of dynamic environments. At the time of this writing, the authors have been unable to devise a general memory update scheme for dynamic environments that avoids having to render the scene twice per character. Such a scheme would certainly be very useful in improving the efficiency of the synthetic vision module, and allowing more characters to be run simultaneously. Other possibilities include invoking the vision module every other frame, or at some regular interval. This could allow one to gain speed (or additional characters) at the expense of accuracy. Clearly, many challenging research issues must be faced before more interesting motions and intelligent behaviors for autonomous animated characters can be realized.

Acknowledgments

The authors wish to thank the Stanford Computer Graphics Laboratory for allowing them to utilize their rendering hardware and video equipment during these experiments. This research is supported in part by a National Science Foundation Graduate Fellowship in Engineering, and MURI grant DAAH04-96-1-007 (Army).

References

[1] R. C. Arkin. Cooperation without communication: Multi-agent schema based robot navigation. *Journal of Robotic*

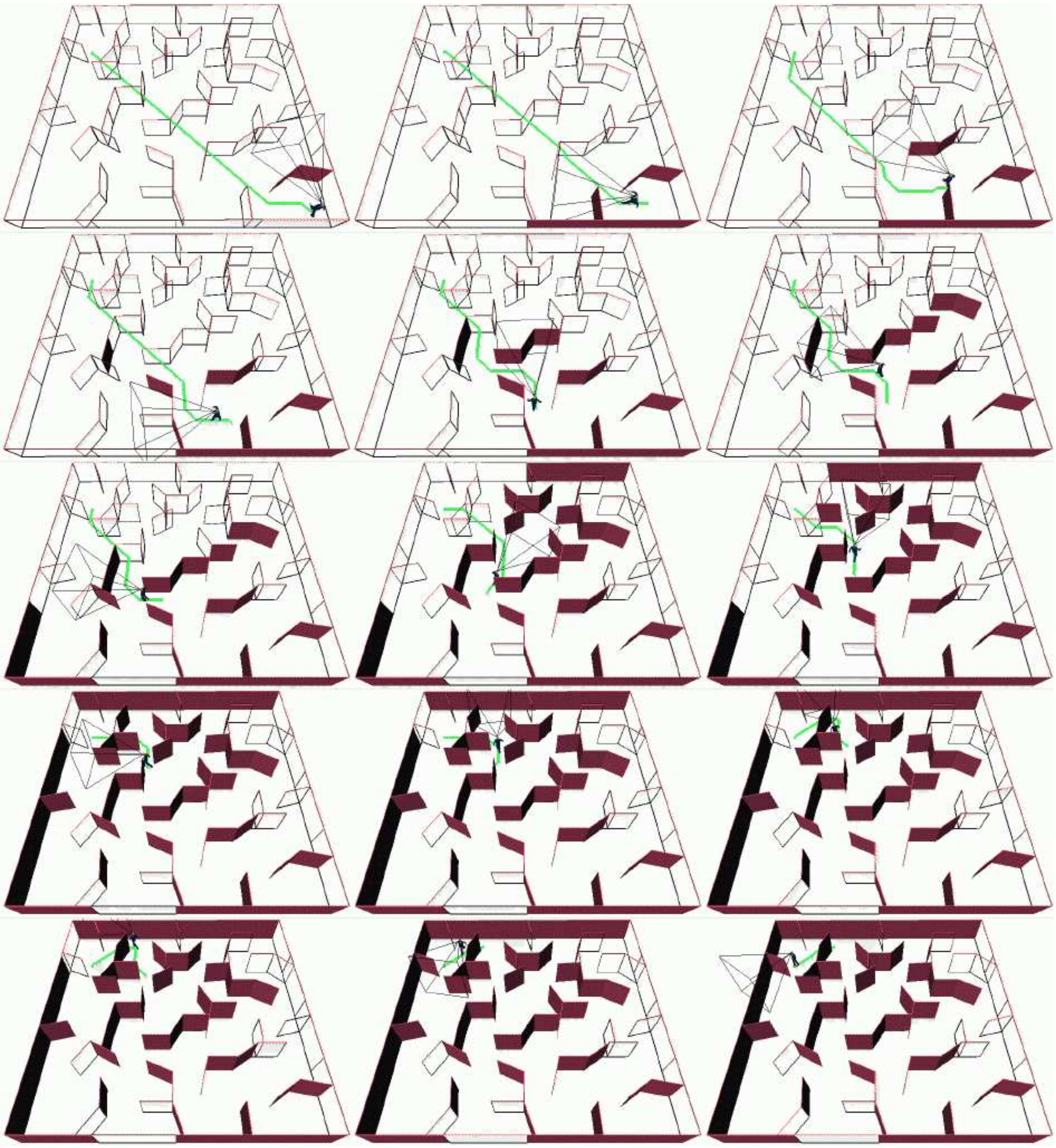

Figure 12. Snapshots of a character exploring an unknown maze environent. Objects rendered solid are those contained in $\mathcal{M}$, while those rendered wireframe are unknown. The top-left image shows the inital frame with the character given the task of navigating from the bottom-right corner to the top-left corner of the maze. A portion of the character's viewing frustum, along with the current path is also shown.

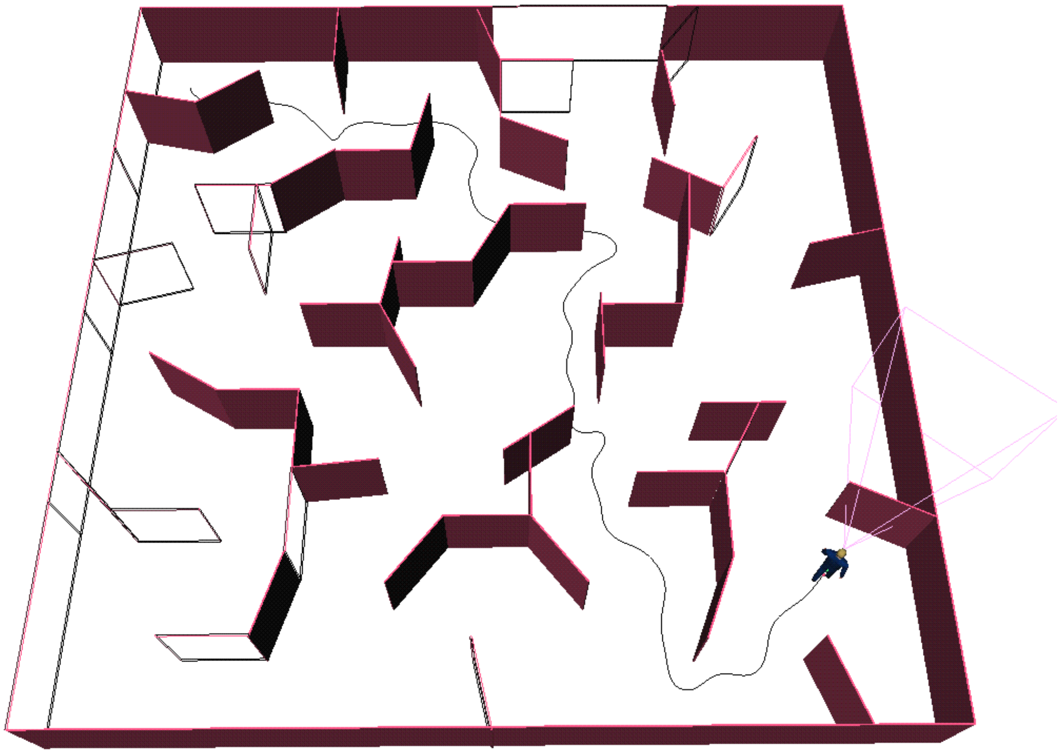

Figure 13. Detail of the trajectory followed by the character during the exploration of the unknown maze environment of Figure 12.

Systems, pages 351–364, 1992.

[2] N. Badler. Real-time virtual humans. *Pacific Graphics*, 1997.

[3] J. Bates, A. B. Loyall, and W. S. Reilly. An architecture for action, emotion, and social behavior. In *Artificial Social Systems : Proc of 4th European Wkshp on Modeling Autonomous Agents in a Multi-Agent World*. Springer-Verlag, 1994.

[4] B. M. Blumberg. *Old Tricks, New Dogs : Ethology and Interactive Creatures*. PhD thesis, MIT Media Laboratory, Boston, MA, 1996.

[5] B. M. Blumberg and T. A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In R. Cook, editor, *Proc. SIGGRAPH '95*, Annual Conference Series, pages 47–54. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[6] D. C. Brogan and J. K. Hodgins. Group behaviors with significant dynamics. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995.

[7] R. A. Brooks. A layered intelligent control system for a mobile robot. In *Robotics Research The Third International Symposium*, pages 365–372. MIT Press, Cambridge, MA, 1985.

[8] J. P. Granieri, W. Becket, B. D. Reich, J. Crabtree, and N. L. Badler. Behavioral control for real-time simulated human agents. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 173–180. ACM SIGGRAPH, Apr. 1995. ISBN 0-89791-736-7.

[9] J. K. Hodgins. Three-dimensional human running. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 1996.

[10] M. R. Jung, N. Badler, and T. Noma. Animated human agents with motion planning capability for 3D-space postural goals. *The Journal of Visualization and Computer Animation*, 5(4):225–246, October 1994.

[11] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning motions with intentions. In *Proc. SIGGRAPH '94*, pages 395–408, 1994.

[12] E. Kokkevis, D. Metaxas, and N. I. Badler. Autonomous animation and control of four-legged animals. In W. A. Davis and P. Prusinkiewicz, editors, *Graphics Interface '95*, pages 10–17. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1995. ISBN 0-9695338-4-5.

[13] J. J. Kuffner, Jr. Goal-directed navigation for animated characters using real-time path planning and control. In *Proc. of CAPTECH '98 : Workshop on Modelling and Motion Cap-*

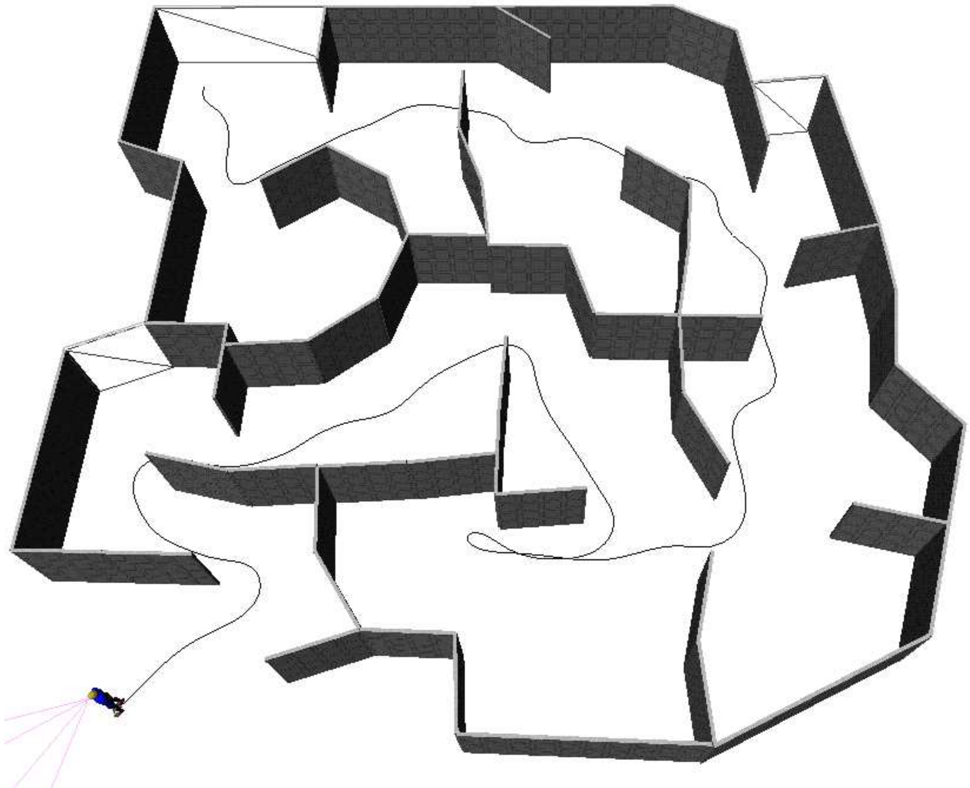

Figure 14. The trajectory followed during exploration of another unknown maze environment.

ture Techniques for Virtual Environments. Springer-Verlag, November 1998.

[14] J. J. Kuffner Jr. *Designing Autonomous Agents for Real-Time Animation (working title)*. PhD thesis, Stanford University *(in preparation)*.

[15] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[16] P. Maes, D. Trevor, B. Blumberg, and A. Pentland. The ALIVE system full-body interaction with autonomous agents. In *Computer Animation '95*, Apr. 1995.

[17] H. Noser, O. Renault, D. Thalmann, and N. Magnenat Thalmann. Navigation for digital actors based on synthetic vision, memory and learning. *Comput. Graphics*, 19:7–19, 1995.

[18] H. Noser and D. Thalmann. Synthetic vision and audition for digital actors. In *Proc. Eurographics '95*, 1995.

[19] K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995. ISSN 1077-2626.

[20] K. Perlin and A. Goldberg. IMPROV: A system for scripting interactive actors in virtual worlds. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, Annual Conference Series, pages 205–216. ACM SIGGRAPH, Addison Wesley, 1996.

[21] O. renault, N. M. Thalmann, and D. Thalmann. A vision-based approach to behavioral animation. *Visualization and Computer Animation*, 1:18–21, 1990.

[22] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[23] G. Ridsdale, S. Hewitt, and T. W. Calvert. The interactive specification of human animation. In M. Green, editor, *Proc. of Graphics Interface '86*, pages 121–130, May 1986.

[24] S. Strassmann. *Desktop Theater: Automating the Generation of Expressive Animation.* PhD thesis, M.I.T. Media Arts and Sciences Program, Toronto, Canada, 1991.

[25] S. Strassmann. Semi-autonomous animated actors. In *Proc. AAAI '94*, pages 128–134, 1994.

[26] D. Terzopoulos and T. Rabie. Animat vision: Active vision in artificial animals. In *Proc. Fifth Int. Conf. on Computer Vision (ICCV'95)*, pages 801–808, Cambridge, MA, June 1995.

[27] D. Thalmann, H. Noser, and Z. Huang. *Interactive Animation*, chapter 11, pages 263–291. Springer-Verlag, 1996.

[28] X. Tu. *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior*. PhD thesis, University of Toronto, Toronto, Canada, 1996.

[29] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In A. Glassner, editor, *Proc. SIGGRAPH '94*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.