Sensor Based Synthetic Actors in a Tennis Game Simulation

Hansrudi NOSER
Daniel THALMANN
Computer Graphics Laboratory,
Swiss Federal Institute of Technology
CH-1015 Lausanne, SWITZERLAND
(noser@lig.di.epfl.ch; thalmann@lig.di.epfl.ch)

Abstract

In this article we propose a model of a tennis game simulation with synthetic actors as players and a referee. The behavior of these actors is based on their synthetic vision and audition. Physical modeling of the ball dynamic and sound rendering enhance realism. An interactive user can also play against a synthetic actor by using a Space Ball.

1. Introduction

In this paper we present a virtual 3D tennis game simulation including synthetic and interactive players and a synthetic referee judging the game. The behavior of the synthetic actors is based on their synthetic vision [RTT90, NRT95] and audition [NT95B]. A physical force field modeling of the environment allows tactile interaction with the environment. The whole system can be run in an interactive mode where a user can play against a synthetic actor. In the simulation mode actors play against each other and a synchronized and rendered sound track can be automatically generated for a raytraced video sequence.

We focus on sensor based behavior modeling allowing autonomous actors to play tennis and to judge a tennis game. Our behavioral L-system [NT93, NT94C, NT96, PRUS90] is extended with new roles for synthetic actors. In section 3 we describe the modeling of various actor sensors (vision, audition, touch) for the perception of the virtual world. Especially the integration of an audition interface

constitutes a new element for actor communication. Then, we present in section 4 some behaviors which are based on the synthetic sensors. We emphasize the behavior control and some particular extensions of the game as the player's strategy, its adaptive reaction to perturbation of the ball trajectory by variable conditions and the path planning during the game. We also detail the behaviors of the players, the referee and the interactive player who can take over the role of a synthetic actor.

In the next paragraphs we describe some previous work on synthetic vision, audition and behavioral animation related to our work. Reynolds [R88] proposed Z-buffer images for collision detection. Jolion [JO94] gives a general overview of methodologies for computer vision. Tu, Terzopoulos and Rabie introduced in [TTA94, TTP94, TR95] a vision-based perception for fishes. Roth-Tabak et Jain [ROT89] introduced an algorithm using dense range data to generate, refine and update a 3D environment model. These data can be used by an autonomous system for navigation and path planning as well as object recognition and manipulation. Renault et al. [RTT90] introduced the concept of synthetic vision as a main information channel between the virtual environment and the digital actor. In [BG95] Blumberg et al. discuss the problem of building autonomous animated creatures which use also synthetic vision.

Real life is full of different sounds. Therefore, its use in video productions can considerably increase the quality of the final product. According to E. M. Wenzel [W92] "the function of the ears is to point the eyes". Durand R. Begault [DB94] published an introductory book on 3D sound for virtual reality and multimedia. It includes descriptions of reverberation modeling and auralization for acoustical design applications. A general methodology to produce synchronized sound tracks for animation is described by Tapio Takala and James Hahn [TAKHA92].

2. The virtual environment

The environment of the actors is modeled with behavioral L-systems which are timed production systems designed to model the development and behavior of static objects, plant like objects and autonomous creatures. The behavioral L-system we use, is based on a timed, parametric, stochastic and conditional production system, force fields, synthetic vision and audition. More information can be

found in [NT93, NT94c, NT95b, NT96]. The L-system interpreter controls and synchronizes the animation of the actors and the virtual environment that integrates geometric, physical and acoustic elements.

2.1 Geometric modeling

The L-system model associates to its symbols basic geometric primitives as cubes, spheres, trunks, cylinders, line segments, pyramids and imported triangulated surfaces. We define the non generic environment as the ground, the tennis court, or walls directly in the axiom of the production system. The generic parts, as growing plants, are defined by production rules. The actors are represented by a special symbol. Their geometric representation can vary form simple primitives like some cubes and spheres, over a more complicated skeleton structure to a fully deformed triangulated body surface usable in a raytracer. The choice of an actor's shape depends on the purpose of the application which can range from tests to interactive real time simulations or raytraced video productions. Actually, the players are represented by simple rackets and the referee by a humanoid actor. The integration of networked articulated actors was the topic of an other publication [NPCTT96].

2.2 Physical modeling

To model realistic virtual environments we use particle dynamics in force fields based on Newton's equation of movement. Particle dynamics can be used to model behavioral animation [NT93, REY87] and for physical simulation. In a force field animation system the 3D world has to be modeled not only geometrically, but also by force fields. Some objects exert repulsion forces on others in case of collisions. Other objects are attractive. There can exist objects being attractive at far distances and repulsive at short distances. Space force fields like gravity or wind can influence trajectories of moving particles. In the tennis game simulation the particle dynamics serves to animate the ball. With this approach the ball movement in gravitation and wind force fields, including collisions of the ball with the ground, the net and the racket can be simulated. In our dynamic particle system each particle is treated as a point object which can carry a vector force field influencing other objects.

The racket is represented by a disc. Its movement is not determined by dynamics. It moves along the trajectory planned by the corresponding actor. The racket however exerts a force on the ball. To simulate wind any appropriate position and time dependent 3D vector force field can be used. However, if it is too strong and too irregular, the performance of the playing actors suffers.

2.3 Sound modeling

The acoustic environment is composed of sound sources and a propagation medium. The sound sources can produce sound events composed of a sound file name, a position in the world, a type of sound, and a start and an end time of the sound event. Spoken words and sentences, collision sounds or background music represent sound events. The sounds used are read from a library of numeric AIFF files.

Figure 1 shows the architecture of our sound environment model. The propagation medium corresponds to the sound event handler which controls the sound events and transmits the sounds to the ears of the actors and/or to a user and/or a sound event track file.

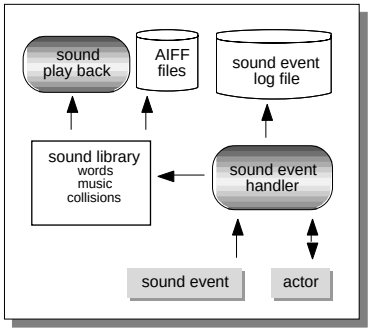


Figure 1. Sound environment architecture

Each sound event is emitted to the sound event handler which controls the playback, the update of the sound event log file, the duration of the sound, and the update of the sound source position during the activation time of the sound. By using the sound event handler, each actor can access the currently active sounds. We do sound rendering only for video production. In this case, at each frame, a sound event log file is additionally updated which is rendered at the end of the animation by a sound renderer described in [NT95b]. For tests or interactive applications we suppose an infinite sound propagation

speed without weakening of the signal. The sound sources are all omni-directional, and the environment is non reverberant.

An acoustic environment can be modeled by using procedures from two modules "sound event handler" and "sound library" shown in Figure 1. The module "sound library" offers the possibility to define a set of sounds and to play them on the SGI hardware. The sound library for the tennis game simulation contains the following prerecorded sounds:

Collision sounds: ball_racket, ball_ground ball_net,

Environment: frog, cricket, music, airplane

Spoken language: advantage, all, deuce, fault, fifteen, forty, game, let, love, server, striker, thirty, blue, green, out, service, left, right, finished.

3. Synthetic sensors of actors

An actor can perceive its geometric, physical and acoustic environment by its synthetic vision, touch and audition sensors. In our implementation of vision-based approach to behavioral animation the synthetic actor perceives its environment from a small window in which the environment is rendered by the computer from the actor's point of view. As the actor can access Z-buffer values of the pixels (distances of the objects' pixels from the observer), the color of the pixels and its own position, it can recognize visible objects and calculate their 3D position in the virtual environment. When visual memory is needed, we model it by a 3D voxel grid realized by an octree data structure, where each pixel of an object, transformed back to 3D world coordinates, occupies a voxel. By comparing at each frame the rendered voxels in the visual field with the corresponding pixel of the vision window, we can update the visual memory by eliminating voxels having disappeared in the 3D world. Thus, the visual memory can reflect the state of a 3D dynamic world perceived by the synthetic actor.

The concept of synthetic vision with a voxelized visual memory is independent of a 3D world modeling. Even fractal objects and procedurally defined and rendered worlds without 3D object database can be perceived, as long as they can be rendered into a Z-buffer based vision window. As a

test-bed for synthetic vision we use the L-system based production system where we don't have any database of the virtual environment which is derived at each frame from the production rules and the current formal symbol string.

In navigation problems all perceived objects are only obstacles. In tennis playing, however, the actors need to differentiate between the ball, the other actor, and the rest of the environment. To get semantic information from vision, we use color coding. By associating a certain color to an object, and giving this knowledge to an actor, it can localize objects by the pixel colors of its vision image.

In the animation system we don't do collision detection explicitly between surfaces. The only thing an actor can "feel" are the force fields. To model a tactile sensor point we create a special particle in the particle system with an appropriate force field. The movement of the particle is not governed by dynamics but related to the actors movement. Through this particle the actor "feels" the forces. Collisions with heavy objects represent strong forces. With the particle's force field the actor can manipulate other objects submitted to particle dynamics.

The audition sensor, or the “ear” of an actor, corresponds to the table of the currently active sounds provided by the sound event handler representing the propagation medium described in the previous section. From this table the actor gets the complete information of each sound event as the identifier, the source, and the position. The semantic information of a special sound identifier is known to the actor and used in the corresponding behavior automaton.

4. Actor behaviors

To model the special behaviors and roles of the actors, necessary for the tennis game, we use an automata approach. Each actor has an internal state which can change each time step according to the currently active automaton and its sensorial input. In the following we use behavior and automaton as synonyms.

4.1 Behavior control

To control the high level behavior of an actor we use a stack of automata for each actor. At the beginning of the animation the user provides a sequence of behaviors and pushes them on the actor's

stack. When the current behavior ends, the animation system pops the next behavior from the stack and executes it. This process is repeated until the actor's behavior stack is empty. Some of the behaviors use also this stack, in order to reach sub-goals by pushing itself with the current state on the stack and switching to the new behavior allowing them to reach the sub-goal. When this new behavior has finished, the automaton pops the old interrupted behavior from the stack and continues. This stack based behavior control gives an actor some autonomy as it can create its own sub-goals, when necessary, while executing the original script.

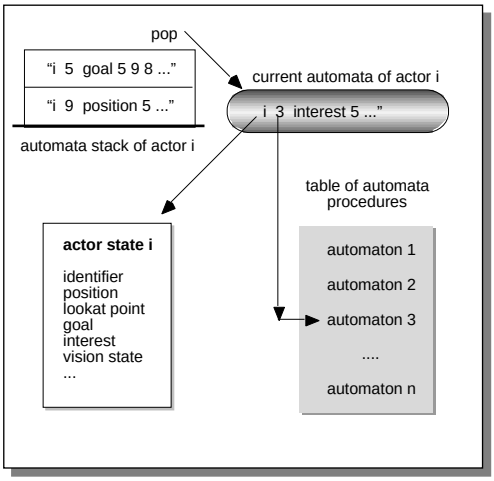


Figure 2. Architecture of the behavior control

We implemented the behavior stack by a stack of command strings. Each string identifies in its header the actor and the automaton. The rest of the string contains commands that modify the actor's state variables. An example is shown in figure 2. If an automaton pops a behavior, it is parsed and interpreted. If a state variable is not affected, it maintains the actual value. The following list shows the behaviors used by the tennis players and the referee.

- **talk:** emits some sounds to the sound event handler
- **listen:** captures some sounds from the sound event handler
- **observe:** observes the environment to update the vision system
- **walk_continuously:** navigation and path planning for actors, based on vision and visual memory

- **tennis_error_detection:** detects errors during a game
- **referee:** judges tennis games and updates a match
- **play_tennis:** plays a tennis game
- **play_tennis_interactively:** the user takes the role of an actor

4.2 Common behaviors

In this section we describe the common behaviors "walk_continuously", "observe", "talk" and "listen" which are used by the referee and the tennis players. When necessary, the referee or the tennis players can pop their actual state on the automata stack, and switch to one of these automata.

Navigation and observe: The “walk_continuously” automaton allows navigation and path planning for actors based on vision and visual memory. It allows an actor to navigate autonomously to a given goal from its actual position. We described the principles of this automaton already in [NRT95].

The “observe” automaton is used by actors which have detected a collision in front of them. The vision system turns around by 360 degrees in order to update the visual memory, and to find visible color coded objects. After having turned around once, the next automaton is popped from the stack.

Talk and listen: A high level behavior, which has to talk at a given moment (see referee), can use the "talk" automaton (see figure 3).

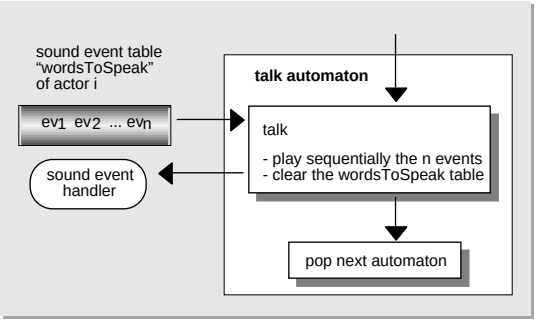


Figure 3: The "talk" automaton

It puts n words into the actor's sound event table "wordsToSpeak", pushes itself on the stack, and changes to the “talk” automaton. The “talk” automaton transmits sequentially these events to the sound event handler by taking into account the duration of each word. When it has transmitted the last event, it clears the table and pops the next automaton from the stack.

The “listen” automaton (see figure 4) permits an actor to concentrate on a sound source and to memorize the next n sound events emitted from this source. It is, for example, used by the "play_tennis" automaton of the synthetic tennis players described later, when it listens to the referee. To simplify the capture of spoken words, we suppose that an actor emits only one sound event at a time. After having captured the n events, the “listen” automaton pops the next automaton from the stack.

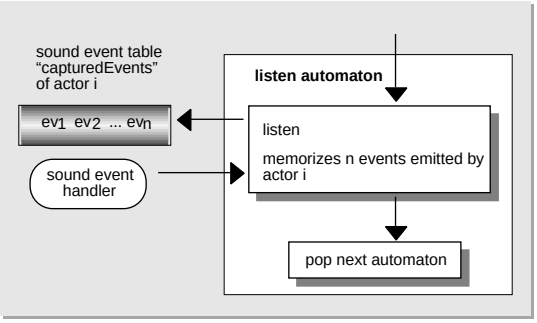


Figure 4. The "listen" automaton

4.3 The referee

The automaton "tennis_error_detection", illustrated in figure 5, permits an actor to track the ball with its vision system, and to detect player faults. It’s actually used by the referee (and in future by other synthetic actors watching a game). This automaton controls the vision system and the audition system of the actor.

The vision system cycles through the states look_around_set, look_around and fix_thing. In the state look_around_set it sets the view angle to 40 degrees and looks for the ball in the state look_around. When it finds the ball, it changes to the state fix_thing where it tracks the ball and adjusts the view

angle (see also the “play_tennis” automaton). When it loses the ball, it goes into the state "look_around_set".

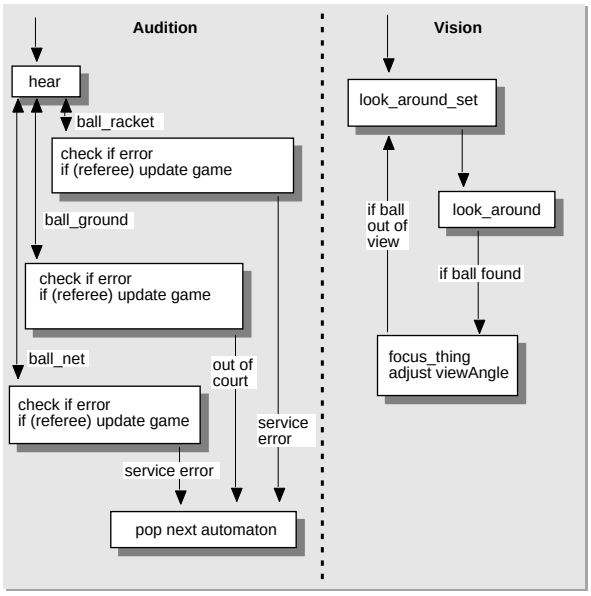


Figure 5. The "tennis_error_detection" automaton

The audition part handles the sound events, controls the game, and if the actor is the referee, it updates also the state of the current game. The audition system has to process the three collision events ball_racket, ball_ground and ball_net. The ball_racket event only produces errors if the striker does not let bounce the ball after the service. Now, the referee updates the game, i.e. the other player becomes the current player, the number of bounces is reset to zero, and if the receiver stroke the ball, the service phase is over.

The "ball_ground" sound event produces an error if the ball is out of the corresponding court side, or if it has bounced twice. If the actor is the referee, the game is also updated, i.e. the number of bounces is incremented. The "ball_net" sound event produces an error only after a service stroke. When one of these errors occurs, the next automaton is popped from the stack.

The “referee” automaton judges and updates a tennis match. The first entry point of this automaton is the state init_tennis_game where the match and the referee are initialized. In the next state "update_tennis_game", the automaton updates the game and the match according to a subset of the

international tennis rules and the error which is responsible for the actual activation of the automaton. For simplicity, we didn't implement the tie break procedure and the change of ends rules.

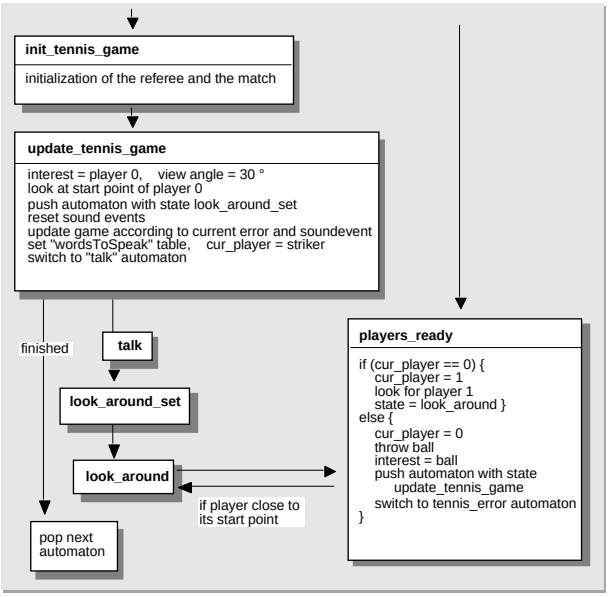


Figure 6. The "referee" automaton

After the update the automaton comments the error, announces the actual score, and indicates the server and the position from where it has to play. Thus, the referee, for example, puts the words "out, fault green, fifteen all, service blue, right" into the table "wordsToSpeak" (see the “talk” automaton), pushes the automaton on the stack with the look_around_set state, and makes the player 0 to its object of interest. Then, it switches to the "talk" automaton which "announces" the words to the other participants of the game

4.4 The players

The evolution of a tennis game can also be described by an automaton with transitions as illustrated in Figure 7. At the beginning the player is in an inactive state. Through the transition t_1 triggered ,for example, through an external event, it changes to the state "at_start" by navigating to its start position. When it arrives there, it goes into the state "look_for_partner", where it waits until its partner has arrived at the start point. It verifies this of course by using synthetic vision. Then, it changes to the state

"follow_ball" where it looks for the ball. When it has seen the ball, it tracks it with the vision system, and estimates its speed and position.

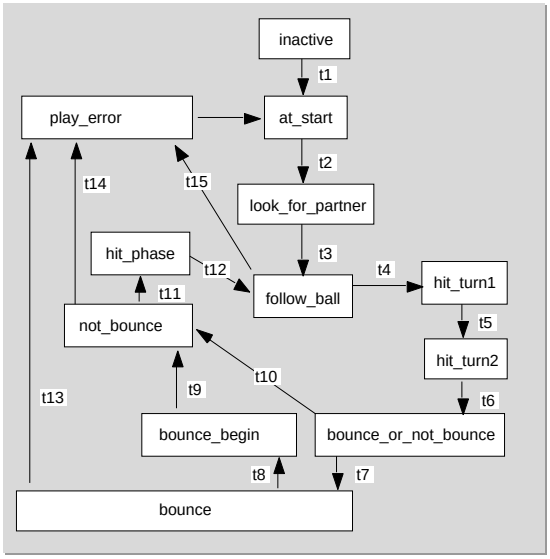


Figure 7. Actor states during a tennis game

Section 3 describes informally how the vision system recognizes the ball in the vision image, and how it estimates its position at a given frame. In order to estimate its velocity, each actor maintains the positions of the last n (=3) frames and derives the velocity from them and the frame to frame time increment. If the actor sees the ball approaching, it goes through transition t_4 into the state "hit_turn_one". As the velocity estimation of the ball is still not satisfactory, it waits some time by changing through the intermediate state "hit_turn_two" to the state "bounce_or_not_bounce", where it has to decide whether to let the ball bounce or not. According to its decision, it changes to the state "bounce" or "not_bounce". In the state "bounce", when the ball is close to the ground, it enters the state "bounce_begin". When the ball has collided and starts mounting, it goes to the state not_bounce. If it is close to the racket ball impact point, it enters into the state "hit_phase" where it strikes the ball according to the game strategy described later. After the stroke the player enters the "follow_ball" state, and a new cycle can begin.

Figure 8 illustrates the different states along the ball trajectory. The numbers one to eleven indicate consecutive events important for player 1. At event 1, the player 1 is in the state "follow_ball" and its

vision system focuses the ball. Some time later, player 0 hits the ball (event 2) and player 1 sees the ball flying back. Now, it passes through the events 3 ("hit_turn_one") and 4 ("hit_turn_two") into the state "bounce_or_not_bounce", where it decides whether to let bounce the ball or not. Between the events 5 and 7 it is in the state "bounce". Event 6 indicates the ball-racket impact position in the case where it lets not bounce the ball. At event 7 it enters the state "bounce_begin". When the ball starts mounting (event 8) after the collision with the ground, the player 1 enters consecutively the states "not_bounce" and "hit_phase" indicated by events 8 and 9.

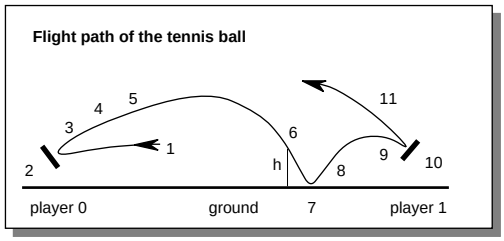


Figure 8: The flight path of the ball.

These last two states are equivalent to the case where it decides not to let bounce the ball, and where it could enter into the state "not_bounce" directly after event 5. Event 10 indicates the hit, and event 11 the state "follow_ball". Now, player one goes back to its start position while continuing to focus the ball. There, it waits until the ball is returned. Then, the whole cycle is repeated.

The game automaton of an actor: During a game the "play_tennis" automaton has to control the actor's vision system, the audition system and its internal state of the game. Each of the three systems has its own state variables which can be mutually manipulated. The audition system control illustrated in figure 9 checks at each frame the sound events. If the automaton detects the sound event "fault", emitted by the referee as described in section 4.3, it pushes the actual "play_tennis" automaton on the stack, together with the necessary initializations allowing a new game. Then, it activates the "walk_continuously" automaton in order to move to its start position on the tennis court.

If the detected sound event is "finished", the whole game is finished, and the actor pops the next automaton from the stack. If it detects the sound event "service", coming from the referee, it pushes the

"play_tennis" automaton on the stack with the initial state "interprete_words", and switches to the "listen" automaton which captures the next two words coming from the referee. When the "listen" automaton has captured the two words, it pops the next automaton, being of course the "play_tennis" one, with the initial state "interprete_words". These two words are the name of the server and the server position “left” or “right”. The actor recognizes whether it is the server or not. If it is the server, it fixes the correct aim point for its service, changes to the "look_around_set" state, and a new game will begin.

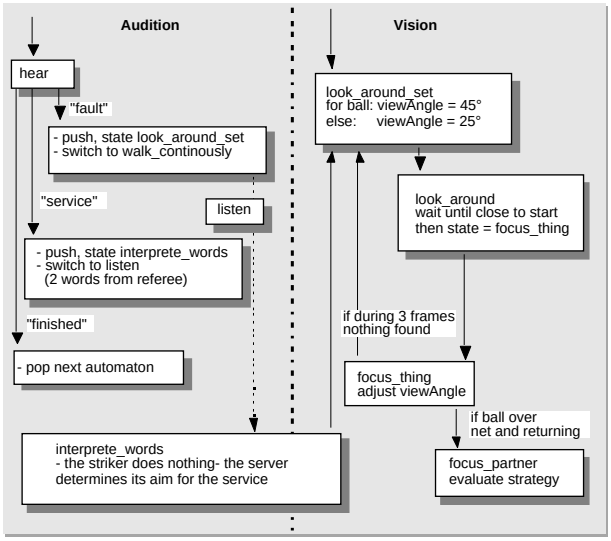


Figure 9: The "play_tennis" automaton

Each actor has a state variable designing its object of interest it is looking for with its vision system. In the state "look_around_set" it initializes its view angle according to the size of its object of interest which can be the ball or its game partner, and it focuses its eyes on the start position of its opponent. Then, it enters the state "look_around". If its object of interest is the ball, it goes to the state "focus_thing" if the ball is visible. If the object of interest, however, is the other player, it waits until it is close to the start point. Only then, it switches to the state "focus_thing". In this state it controls several features. As the name of the state indicates, one task of the actor is to focus the vision system on the object of interest, and to adjust the view angle. It tries to maintain the number of pixels of the object between 5 and 12 by multiplying it by a factor 1.5 or 1/1.5 respectively. The adjustment of the view angle is necessary as the vision window generally is small (30x30 pixels), and small objects like the ball would disappear at long distances. When the object of interest or the actor move fast, it can

happen that the vision system fails to track it. In this case the state is changed to "look_around_set" where the actor starts again to look for the ball.

If during the impact point estimation the actor estimates the impact point to be outside of its court, it decides to call a let, and to move directly back to its start position, waiting there for a new game. That is why, it pushes the "play_tennis" automaton on the stack, together with the initializations for a new game, and activates the "walk_continuously" automaton with its start point as goal. If the vision state is "focus_thing" and the game state "look_for_partner", the state is reset to "look_around_set", and the ball becomes the object of interest. When the object of interest is the ball, and when the vision state is still "focus_thing", the striker tries to determine its opponents position to use it for its game strategy. That's why, when the ball flies over the net, the player changes its vision state to "focus_partner" after having set its object of interest to its opponent.

If the vision state is "focus_partner", the actor evaluates the aim point of its next stroke according to its strategy and the actual position of the opponent. Then, it goes back into the "focus_thing" state, after having selected the ball as object of interest.

Impact point estimation and learning: During the game each actor should estimate the future impact point immediately after the stroke of its opponent. We designed the actor with the capability to learn from experience, and to adapt itself to changing conditions as changing mass of the ball, changing air resistance, changing precision of the numerical integration characteristics, and time varying wind force fields. At each frame the actor extracts the position of the ball from its vision image (see section 3). Every n-th (=3) frame it derives the velocity of the ball according to equation

$$\mathbf{v}(t) = (\mathbf{P}(t) - \mathbf{P}(t - n * t\_interval)) / n / t\_interval \quad (1)$$

From this current velocity and the current position of the ball it can calculate the future impact point and impact time. We suppose, that the actor wants to hit the ball at a certain height h. As the vertical velocity component of the ball is in general not too big, we can neglect air damping and assume the ball moving in vertical direction according to equation 2.

$$m\ \dot{v} = m\ g$$
$$x(t) = v_0\ t + 0.5\ g\ t^2 \qquad (2)$$

with m: mass of the ball

g: gravity acceleration

v_0: initial vertical speed of the ball

t: time

The impact time at height h can easily be calculated from the quadratic equation. To estimate the horizontal x and z components of the impact position we chose a heuristic quadratic function where some coefficients are adjusted during the game according the actors experience. This heuristic function should be linear at short distances and reduce the extrapolated distance d(t) of equation (3) at long distances. We chose the distance function shown in equation (4). The future impact time t_i and the initial velocity v(t) are known at a given time t. The variables x and o model the function's linearity at short distances, and the reduction of the extrapolated distance at long distances. Figure 10 shows this function with the values x=2 and o=10000.

$$d(t) = v(t)(t_i - t) \qquad (3)$$

$$dist(t) = d(t)*(d(t)^2 + o) / (x * d(t)^2 + o) \qquad (4)$$

d(t): extrapolated distance

t_i: future impact time

t: actual time

dist(t): corrected distance

x, o: modeling variables

Each actor estimates the future impact point every n-th (=4) frame when it is in a state between "bounce_or_not_bounce" and "hit_phase". If the ball is close enough to the actor, the estimation can be

stopped as the impact point is precise enough. Then, the actor enters the "hit_phase". Thus, at long distances, the impact point estimation is rough, and it becomes sufficiently exact at short distances to execute the stroke. The variable o = 10000 is a heuristic constant value. The value of x, however, is adjusted by the player during the game. This variable permits to improve the estimation of the impact point at long distances. The factor x is determined by the system of equation (5).

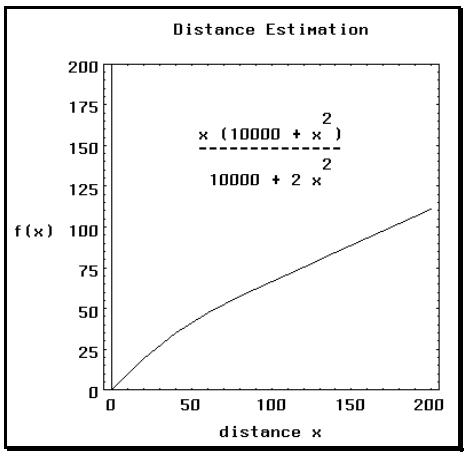


Figure 10. The heuristic function correcting the extrapolated distance.

$$x = (d(d^2 + o) / dist - o) / d^2$$

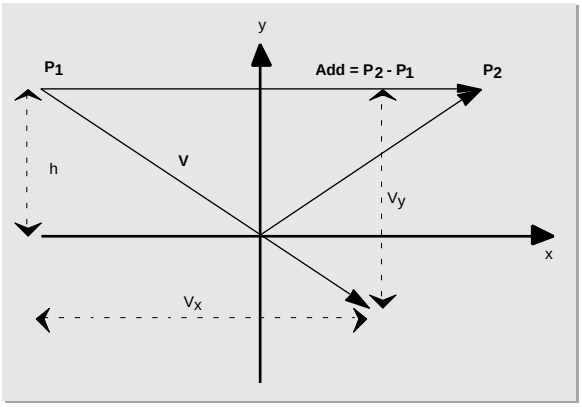
$$d = v_x(t_1) * (t_2 - t_1)$$

$$dist = \mathbf{P_2}(t_2) - \mathbf{P_1}(t_1) \qquad (5)$$

During the game, when the actor enters the "bounce_or_not_bounce" state, it memorizes at time t_1 the speed **v** and the position $\mathbf{P_1}$ of the ball. At time t_2, when it enters the state "hit_phase" or "bounce_begin", it memorizes the position $\mathbf{P_2}(t_2)$ of the ball. Now, it can determine a new value of x. As final value of x it calculates the average of two consecutive values. Thus, each actor adapts itself to the current conditions, which is useful if a wind, for example, disturbs the ball movement at long distances.

Impact point determination, game strategy and stroke planning: During the game a player has to decide whether to let bounce the ball or not. There exist two possible impact points. The first one, $\mathbf{P_1}$, is

estimated according to the previous section before bouncing. The second one, **P_2,** after bouncing, can be approximated according to equation 6 and figure 11. The ball velocity at **P_1** is approximated with the actual ball speed. This value is of course too big, but as only a rough approximation is needed at long distances, it doesn't matter, and at short distances the approximation is good enough. Now, the actor can choose as future impact point the one which is closest to its actual position, and, according to its decision, it enters the "bounce or not_bounce" state.



P_1: first possible impact point

P_2: second possible impact point

h: height of impact point

V: speed of arriving ball at point **P_1**

Figure 11. Estimation of the second impact point P_2

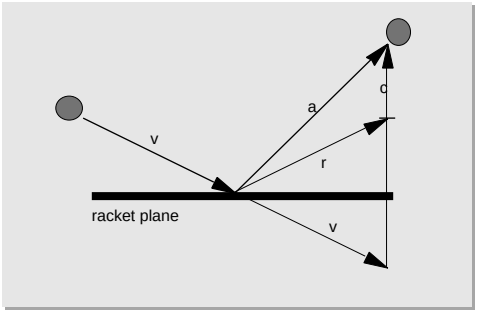
The actors perform a simple game strategy. When the ball flies over the net, the receiver glances during one frame to its opponent's side. If it can localize the ball, it chooses the from its opponent most distant corner of the court area as aim point for its next shot. Next, the receiver has to determine the speed and the direction of its racket at the moment of the stroke.

$$\vec{P}_2 = \vec{P}_1 + a \begin{bmatrix} \frac{V_x}{|V_y|} h \\ 0 \\ \frac{V_z}{|V_y|} h \end{bmatrix} \qquad a = 2 \tag{6}$$

This situation is illustrated in figure 12. **v** is the incoming ball speed and **c** the racket velocity at collision time. The racket surface is always perpendicular to its velocity. As the racket moves along a spline, its mass can be supposed to be infinite, and the ball reflection becomes purely geometrical. The vector **a** is the wished velocity of the ball after the collision in order to hit the aimed point. This point determines the horizontal direction of the vector **a**. To simplify the strategy, we fix the vertical inclination of vector **a**. As a consequence, we can estimate the speed of the resulting ball velocity with a heuristic distance dependent function. The direction of the velocity **a** is easily determined from the estimated impact point, the aim point, and the vertical inclination of the velocity. The speed is approximated by the following heuristic formula

$$\left|\vec{a}\right| = c1 + c2 \left|\vec{p}_{impact} - \vec{p}_{aim}\right| \tag{7}$$

which is proportional to the horizontal distance between the impact point and the aim point. The coefficient c1 is an appropriate constant On the other hand, coefficient c2 is adjusted by the actor itself according to the comparison of the planned aim point and the effective distance traveled by the ball. After the stroke the actor memorizes its impact point and the horizontal distance to the aim point. When it "hears" the ball_racket collision sound of its opponent or the "ball_ground" collision sound, it adjusts the coefficient c2, according to the effective deviation of the effective aim point that it extracts from the vision system.



v: velocity of arriving ball
a: resulting velocity of ball
c: velocity of striking racket
Figure 12. Ball-racket collision

Figure 12 summarizes the geometrical situation of the racket-ball collision. The vector **a** is the wished resulting ball velocity after the hit. The vector **v** is the arriving ball velocity, and **r** is its reflection vector. According to figure 12 the velocity **c** of the racket, necessary to make the ball leaving with velocity **a** after the collision, is determined by the following equations.

$$\mathbf{c} = x*(\mathbf{a} - \mathbf{v}) \quad (8)$$

$$\mathbf{r} = \mathbf{a} - \mathbf{c} \quad (9)$$

The solution

$$x = (\mathbf{a}^2 - \mathbf{v}^2) / (\mathbf{a}-\mathbf{v})^2 \quad (10)$$

allows us to calculate the racket speed at the moment of the impact

The planning of the path of a racket during a game is not a trivial task. When it has to strike the ball, it has to be at a given time at the impact point and to move with a given speed in a given direction in order to correctly strike the ball. In the previous sections we have seen how the impact point, the impact racket velocity, and the impact time are estimated every n-th frame. From this data the actor creates now a timed 3D spline that it will follow. The racket is always parallel to its movement direction.

4.5 The interactive player

If a user wants to play interactively against a synthetic actor, he may use a Space Ball or a racquet equipped with position and orientation sensors, and take the role of one player. In this case the actor is controlled by the "play_tennis_interactively" automaton which maps the 3D Space Ball position and orientation data to the actor's racket. Moreover, the scene is displayed from the actor's point of view to allow the interactive user to see the scene through the actor’s "eyes”. With the Space Ball a user can control the position, the speed and the orientation of the racket. Its position and orientation determine also the position and look at point of the camera of the main display. As the racket and vision system are linked to a repelling force field of the size of the racket, the user can strike the ball. To facilitate the

game the user can choose the resolution of the Space Ball movement, and he can fix its height at a given value. A beginner can also fix the racket inclination. Through the racket the user is merged into the virtual environment and can be perceived by the other actors as the referee and his game partner. For these other actors, however, nothing has to be changed. They behave in both cases the same way.

During the interactive game the vision system of the manipulated actor is deactivated as it is useless. The "play_tennis_interactively" automaton is very simple as the behavior is now determined by the user. If the game is finished, which is determined by the referee, the next automaton is popped from the stack.

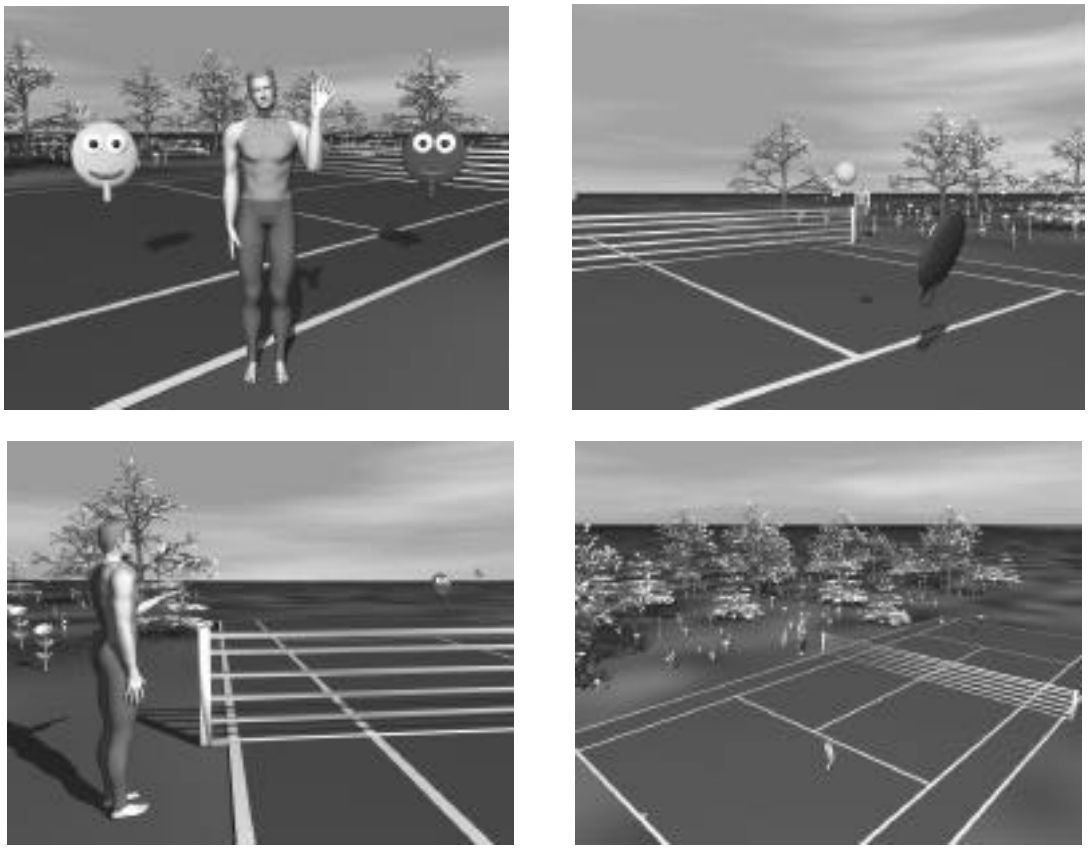


Figure 13. Some images from a raytraced tennis match with synthetic players and a referee

5. Implementation and results

We implemented the L-system based animation system in C/C++ on Silicon Graphics workstations. The interactive game can be played on an Indigo 2 Extreme or on an Onyx without problems in real

time. We produced several raytraced videos with rendered soundtracks. As raytracer we used Rayshade from Craig E. Kolb. Figure 13 shows 4 images from one of the videos where the referee is represented by a synthetic humanoid [BCH95]. The blue and the green racket like actors, and the rest of the environment are modeled by L-systems.

6. Conclusion

We presented a synthetic sensor based tennis match simulation for autonomous players and an autonomous referee, implemented in a L-system based animation system. The different behaviors of the actors are modeled by automata, controlled by an universal stack based control system. As the behaviors are severely based on synthetic sensors, being the main channels of information capture from the virtual environment, we obtain a natural behavior which is mostly independent of the environment representation. By using this sensor based concept the difference between a digital actor and an interactive user becomes small. Therefore, both can be easily exchanged as demonstrated with the interactive game facility.

Acknowledgments

This research was partially sponsored by the Swiss National Foundation for Scientific Research.

Literature

[BCH95] Boulic R., Capin T., Huang Z., Kalra P., Lintermann B., Magnenat-Thalmann N., Moccozet L., Molet T., Pandzic I., Saar K., Schmitt A., Shen J., Thalmann D., "The Humanoid Environment for Interactive Animation of Multiple Deformable Human Characters", Proc. Eurographics '95, 1995, pp 337 - 348.

[BG95] B.M. Blumberg, T.A. Galyean, *Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments*, SIGGRAPH 95, Conference Proceedings, August 6-11, 1995, ACM Press, pp. 47-54.

[DB94] Durand R. Begault, 3D-Sound for Virtual Reality and Multimedia, AP Professional, 1994.

[JO94]Jean-Michel Jolion, *Computer Vision Methodologies*, CVGIP: Image Understanding, Vol. 59, No 1, January, pp 53-71, 1994

[NPCTT96] H. Noser, I. S. Pandzic, T. K. Capin, N. M. Thalmann, D. Thalmann, *Playing Games through the Virtual Life Network*, ALIFE V, Oral Presentations, May 16-18, 1996, Nara, Japan, pp. 114-121

[NRT95] Noser H., Renault O., Thalmann D., *Navigation for Digital Actors Based on Synthetic Vision, Memory, and Learning*. Comput. & Graphics, Vol. 19, No. 1, pp 7-19, 1995

[NT93] Noser H. Thalmann D., *L-System-Based Behavioral Animation*, Proceedings of the First Pacific Conference on Computer Graphics and Applications, Pacific Graphics 93, Aug. 1993, World Scientific Publishing Co Pte Ltd, pp. 133-146

[NT94c] H. Noser, D. Thalmann, Artificial Live and Virtual Reality, Chapter: *Simulating Life of Virtual Plants, Fishes and Butterflies* edited by Nadia Magnenat Thalmann and Daniel Thalmann, 1994 John Wiley & Sons, Ltd.

[NT95b] Noser H., Thalmann D., S*ynthetic Vision and Audition for Digital Actors*, Computer Graphics forum, Vol. 14. Number 3, Conference Issue, Maastricht, The Netherlands, pp. 325 -336, August 28 - Sept. 1, 1995

[NT96] H. Noser, D. Thalmann, *The Animation of Autonomous Actors Based on Production Rules*, Proceedings Computer Animation'96, June 3-4, 1996, Geneva Switzerland, IEEE Computer Society Press, Los Alamitos, California, pp 47-57

[PRUS90] P. Prusinkiewicz, A. Lindenmaer, *The Algorithmic Beauty of Plants* (1990), Springer Verlag

[R88] Craig W. Reynolds, *Not Bumping into Things*, SIGGRAPH course 27, notes: Developments in Physically-Based Modeling, 1988, G1-G13

[REY87] Reynolds C (1987), *Flocks, Herds, and Schools: A Distributed Behavioral Model,*Proc. SIGGRAPH 1987, Computer Graphics, Vol.21, No4, pp.25-34

[REY93] C.W. Reynolds, *An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion*, in: Meyer JA et al. (eds) *From Animals to Animats*, Proc. 2nd International Conf. on Simulation of Adaptive Behavior, MIT Press, 1993.

[ROT89] Y. Roth-Tabak, R. Jain, *Building an Environment Model Using Depth Information*, Computer, June 1989, pp 85-90

[RTT90] O. Renault, N.M. Thalmann, D. Thalmann (1990), *A Vision-based Approach to Behavioral Animation*, The Journal of Visualization and Computer Animation, Vol 1, No 1, pp 18-21

[TAKHA92] Tapio Takala, James Hahn, *Sound Rendering*, Computer Graphics, proceedings, SIGGRAPH '92, Vol 26, No 2, July 92, ACM Press

[TR95] Demetri Terzopoulos, Tamer F. Rabie, *Animat Vision: Active Vision in Artificial Animals* , Proc. of the Fifth Int. Conf. on Computer Vision (ICCV'95), Cambridge, MA, USA, June, 1995, 801-808.

[TTA94] X. Tu and D. Terzopoulos, *Artificial Fishes: Physics, Locomotion, Perception, Behavior*, Proc. SIGGRAPH '94, Computer Graphics, pp.42-48.d

[TTP94] X. Tu and D. Terzopoulos, *Perceptual Modeling for the Behavioral Animation of Fishes*, Proc. Pacific Graphics '94, World Scientific Publishers, Singapore, pp.165-178

[W92] Wenzel, E. M., "*Localization in Virtual Acoustic Displays*", PRESENCE: Volume 1, Number 1 (1992), pp. 80-107