

Generic 3D Ball Animation Model for Networked Interactive VR Environments

Hansrudi Noser¹, Christian Stern¹, Peter Stucki¹, Daniel Thalmann²

¹Institut für Informatik der Universität Zürich, Winterthurerstr. 190
CH-8050 Zürich, Switzerland

{noser, stern, stucki}@ifi.unizh.ch

²Computer Graphics Lab, Swiss Federal Institute of Technology
CH-1015 Lausanne, Switzerland
thalmann@lig.di.epfl.ch

Abstract. This paper describes a versatile, robust, and parametric ball animation model that can be used in many types of interactive Virtual Reality (VR) environments. The generic model is particularly useful for animation of ball-like objects such as tennis balls, footballs, or darts, and for networked collaborative environments where low frame rates, network delays and information losses complicate collision detection of fast moving objects. The ball animation model includes a multi-level physical modeling as well as collision detection and treatment. Finally, the parameterization of typical applications is discussed.

1 Introduction

In most cultures many popular ball games exist. Therefore, their use in VR applications is of economic interest. The general problem of ball animation in distributed VR environments is not extensively treated in literature. In [1, 2, 3] tennis game modeling with autonomous actors and interactive users is presented. The authors used a physics-based particle system approach for the ball animation. In [4] a networked VR application of a tennis game is described, where parts of the ball-model described in this work have been used. However, the ball-model itself is described only marginally.

Particle dynamics, as it is used for ball animation, is extensively treated in literature. Differential equations of particles and their analytic solution can be found in [5]. Similarly, there exist many books on collision detection. Our work is inspired from a method described in [6].

The main problems of ball physics such as dynamics as well as collision detection, and treatment are solved. However, ball animation in distributed VR environments implies some VR specific problems due to real-time constraints, interactivity, as well as update-delays and possible update-losses.

In general animated balls are fast moving objects colliding with static obstacles – such as the floor, nets, or walls – or with other fast moving objects such as feet, fists, or rackets. As in distributed environments frame rates and update rates of the position of balls and interactively moved objects can be low (~6 – 15 Hz), well defined

trajectories of the objects are missing making it difficult to calculate collisions and collision responses. In such cases force field based ball models with numerical integration of the differential equations [2] will fail, or considerably slow down the speed of the animation process.

Another important problem for an interactive player in VR is to hit the ball, as in general his radius of action is limited, and the exact 3D localization of the ball depends strongly on the quality of stereo-view, 3D immersion, and animation rate.

All these limiting factors of a distributed VR environment are requiring a specialized ball animation model. In this paper we propose a generic ball animation model including multi-level physics as well as collision detection and treatment, optimized for a networked VR environment, where the ball model can be adapted to VR display quality, network speed, game type, and game levels.

2 Physical Model

In this article we describe some basic elements of a virtual environment for games with ball-like objects, the floor, a net, and racket-like objects that can hit the ball. In a VR application these elements can be combined easily to a variety of interactive ball games.

We suppose interactive users or autonomous actors to animate the racket-like objects and trying to hit a ball. In order to simplify certain calculations, we can fix a coordinate system with gravity acting in y-down direction and a flat ground aligned to x, z coordinate axes with the origin at a court edge. The net is supposed to be vertical and parallel to the z-axes. Most ball game environments can easily be mapped to this orientation by using elementary transformations.

Our model approximates the ball movement by simplified physical laws. In most games the vertical ball movements are in general slow compared to the horizontal movement. Therefore, we can neglect the air friction term for the vertical ball movement. This approach simplifies certain subsequent calculations. The differential equation is given by equation 1.

m:	mass of ball	\vec{g} :	gravity acceleration
\vec{Y} :	vertical ball position	t:	time

$$m \cdot \ddot{\vec{Y}} = -m \cdot \vec{g} \quad (1)$$

Air resistance, however, which is proportional to the horizontal velocity, influences the horizontal ball movement. This term is necessary for a natural looking ball animation. The corresponding differential equation is given by equation 2.

\vec{X} :	horizontal position of the ball	β:	air friction
\vec{V} :	horizontal velocity of the ball	\vec{V}_o :	initial velocity of the ball

$$m \cdot \ddot{\vec{X}} = -\beta \cdot \dot{\vec{X}} = -\beta \cdot \vec{V}, \quad b = \frac{m}{\beta} \quad (2)$$

Finally, the complete ball movement is governed by the superposition of the horizontal and vertical movement. The analytic solution of the above differential equations 1 and 2 is given by equation 3.

$$\dot{\vec{P}}(t) = \begin{pmatrix} V_{ox} \cdot e^{-t/b} \\ V_{oy} - gt \\ V_{oz} \cdot e^{-t/b} \end{pmatrix}, \quad \vec{P}(t) = \begin{pmatrix} X_0 + V_{0x} \cdot (1 - e^{-t/b}) \cdot b \\ Y_0 + V_{0y} \cdot t - 0.5 \cdot g \cdot t^2 \\ Z_0 + V_{0z} \cdot (1 - e^{-t/b}) \cdot b \end{pmatrix} = \vec{X}(t) + \vec{Y}(t) \quad (3)$$

For high level game modes, we also take into account the ball spin. We suppose that the spin damping during the ball flight in the air lasting t seconds is given by equation 4.

$$\text{spin AirDamping} = \text{sDamp} / (\text{sDamp} + t) \quad (4)$$

sDamp: spin damping factor for the ball moving in air.

We use only a simplified physical model for the game, which neglects secondary effects. For example, we ignore the small influence of the ball spin on the ball's trajectory, but on the other hand, we treat the response when bouncing on the floor or when colliding with a racket.

The rackets are important elements of the ball animation model. By hitting the ball, they can modify its trajectory determined by the physical laws. In the computational model the rackets are geometrically represented by discs of a given radius r_{disc} . In the virtual environment, however, they are generally represented by nice looking triangulated complex surfaces. All we have to do for collision detection is to map them to the corresponding discs of the computational model.

As already mentioned, interactive players or autonomous actors determine the racket's position and orientation. 3D tracking devices such as Flock of Birds (FOBs) or SpaceBalls, for example, capture their movement. The resulting information describing these movements is transmitted over the net to the ball server and potential other clients. The ball server memorizes the last n (typically $n=3$) racket positions and orientations in order to estimate, interpolate, or extrapolate actual racket positions, orientations, and velocities that are needed for collision detection.

3 Collision Detection and Response

In our model the ball collides only with the floor, the net and the rackets. Therefore, these are important elements of the ball model. Collision detection with further objects of the virtual environment could be added, but it would slow down the animation speed.

The floor and the net are static objects. Therefore, collision detection with these static objects is relatively simple to simulate. The rackets, however, are dynamic objects with sampled position and orientation. Collision detection between the ball and the rackets need some special consideration and is discussed in the next sections.

Each time a collision event is detected, we first calculate the actual velocity of the ball, which serves as actual velocity value for subsequent collision response calculations. Then we reset the time to zero, and the collision response determines the initial ball position and velocity for the evolution after the collision. After a collision

and at the beginning of the animation the initial ball position and velocity are in general indexed by zero, such as \mathbf{P}_0 or \mathbf{V}_0 . Vectors are indicated by bold letters in the text or by vector signs in equations.

Ball - Floor. The ball - floor collision detection is immediate according to Fig. 1. When the actual position \mathbf{P} of the ball is below floor level ($P_y < \text{floor_level}$), then collision occurs. The new initial position \mathbf{P}_0 after the collision event is simply adjusted to the floor level to avoid floor penetration. This correction simulates a slight gliding of the ball on the floor when bouncing with some additional horizontal speed. The corresponding collision response is given by equation 5.

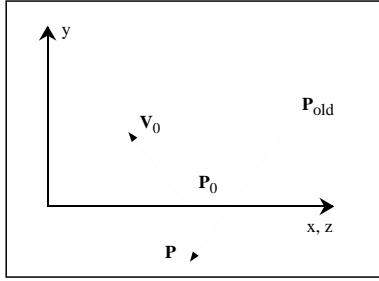


Fig. 1. Ball - floor collision detection

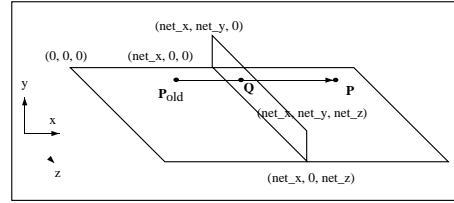


Fig. 2. Ball - net collision detection

$$\begin{aligned}
 f &= e^{-t/b} \\
 \vec{P}_0 &= \begin{pmatrix} P_x \\ \text{floor_level} \\ P_z \end{pmatrix} : \text{new initial position} \\
 \vec{V}_0 &= \begin{pmatrix} V_{0x} \cdot f \cdot \text{floorDamping} \\ -(V_{0y} - g \cdot t) \cdot \text{floorDamping} \\ V_{0z} \cdot f \cdot \text{floorDamping} \end{pmatrix} : \text{new initial velocity} \\
 t &= 0
 \end{aligned} \tag{5}$$

The ball-floor collision event provokes a collision sound if $V_{0y} > \text{sound_threshold}$. The sound feed back for indicating collisions is an important element for the users' immersion quality. Each time a collision is detected, the ball server broadcasts a corresponding sound event to all participating clients.

In equation 5, the vertical speed is set to zero if $V_{0y} > \text{bounce_threshold}$. This last correction avoids infinite small bounce movements of the ball that are due to the gravitation force acting constantly to the ball and pushing it repeatedly into the floor. The floorDamping factor simulates a loss of energy at each bounce on the floor. It can be used to model floor and ball properties.

Ball - Net. Ball-net or ball-wall collisions represent another important class of collision events. We treat only the ball-net case as ball-wall collisions can be treated with only minor modifications.

The collision point \mathbf{Q} of the line through the actual ball position \mathbf{P} at time t and the previous one \mathbf{P}_{old} at time $t-dt$ is given by the pseudo code of **Fig. 3** that refers to Fig. 2. The corresponding response of a ball-net collision is given by equation 6.

$$\begin{array}{l}
 \text{if } (P_{oldx} = P_x) \text{ then } \mathbf{no\ collision} \text{ as line segment and net plane parallel} \\
 \text{else } \vec{Q} = \vec{P}_{old} + s \cdot (\vec{P} - \vec{P}_{old}) \\
 \quad Q_x = net_x = P_{oldx} + s \cdot (P_x - P_{oldx}) \\
 \quad \Rightarrow s = \frac{net_x - P_{oldx}}{P_x - P_{oldx}} \\
 \text{if } s \notin [0..1] \text{ then } \mathbf{no\ collision} \text{ as } \vec{Q} \text{ not on line segment } \vec{P}_{old}\vec{P} \\
 \text{else calculate } \vec{Q} = \vec{P}_{old} + s \cdot (\vec{P} - \vec{P}_{old}) \\
 \quad \Rightarrow \mathbf{collision} \text{ if } Q_z \in [0..net_z] \wedge Q_y \in [0..net_y]
 \end{array}$$

Fig. 3. Ball-net collision detection

$$\begin{array}{l}
 f = e^{-t/b} \quad \vec{V}_0 = \begin{pmatrix} -V_{0x} \cdot f \\ V_{0y} - g \cdot t \\ V_{0z} \cdot f \end{pmatrix} \cdot netDamping : \text{new initial velocity} \\
 t = 0 \quad \vec{P}_0 = \begin{pmatrix} V_{0x} \cdot \Delta t / 10 \\ 0 \\ 0 \end{pmatrix} + \vec{Q} : \text{new initial position}
 \end{array} \quad (6)$$

The new initial position is slightly moved in direction of the new velocity after the net collision. This shift avoids multiple consecutive collisions with the net and places the ball slightly outside the net. The *netDamping* factor has typically a value between zero and one. It simulates a loss of energy of the ball when it enters into collision with the net. It can be used to model the net properties. Note, that in equation 6 the time t represents the time passed since the previous collision event.

Ball - racket. Ball-racket collisions are the most critical ones. In general both objects are moving fast. Additionally, the racket trajectories are mostly represented by sampled noisy data produced by 3D tracking devices and transmitted with certain delays at low sampling rates (typically 2 to 12 frames per second) through the network.

Simple collision detection, for instance, can be realized by calculating the distance of the ball from the racket center. If the ball is closer than a certain collision distance, we are assuming a collision. This collision detection method is not precise, but it is very robust and could be used in a low-level game mode, or in cases, where the racket data is very noisy at low frame rates. In this case the ball position \mathbf{P} at time t is the

collision point Q . This approach, however, detects not all ball-racket collisions. If the ball and the racket move so fast that the distances traveled between two consecutive frames is bigger than the collision distance d_{COLL} , a potential collision is ignored.

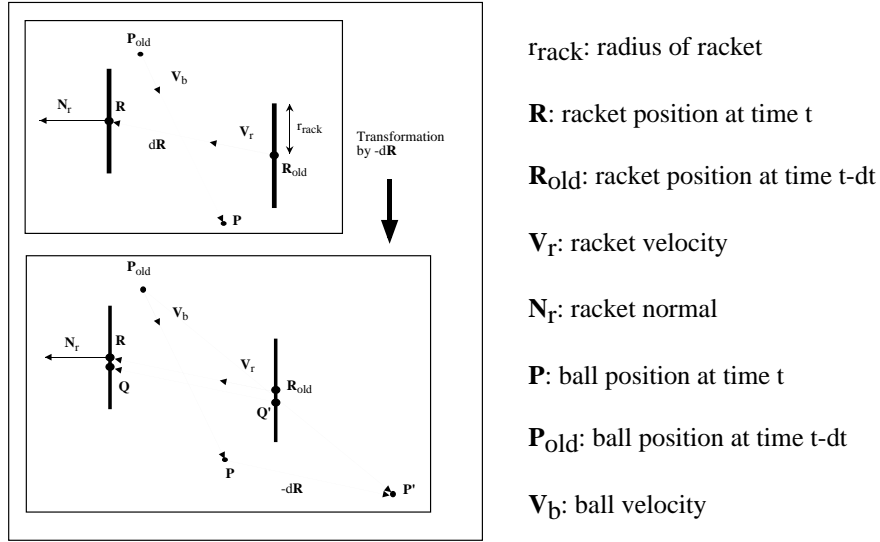


Fig. 4. Ball - racket collision detection using a disc like racket surface

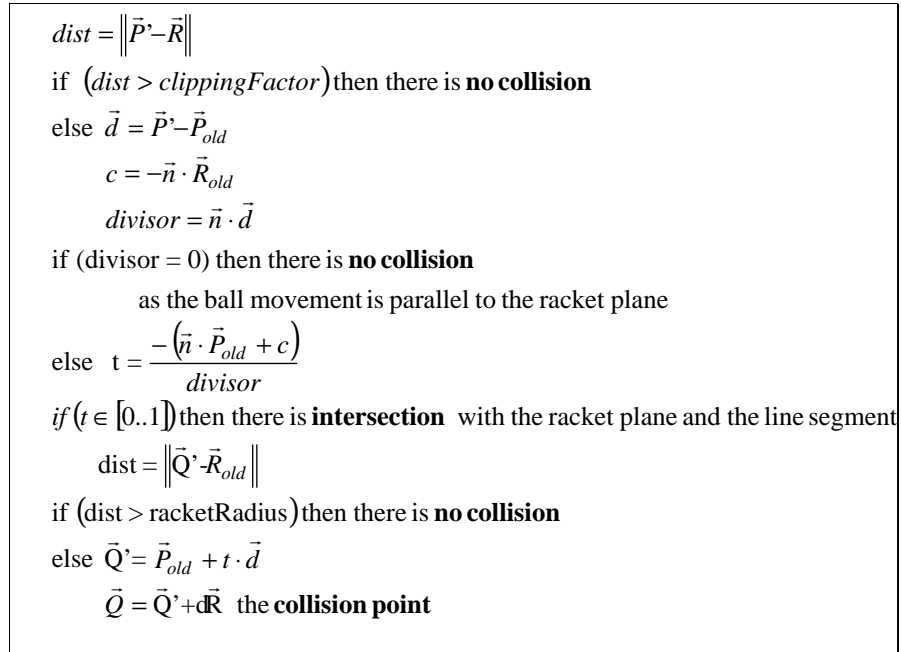


Fig. 5. Ball-racket collision detection

A more precise collision detection method is presented now. In a first step, the actual ball position \mathbf{P} is transformed into the moving coordinate system of the racket where the collision detection can be calculated such as illustrated in **Fig. 4**. We use two consecutive ball and racket positions of two consecutive frames. The transformed ball position \mathbf{P}' is shifted by the negative movement of the racket ($-\mathbf{dR} = \mathbf{R}_{\text{Old}} - \mathbf{R}$). In this coordinate system the racket is static, and therefore, we can now calculate the intersection of the racket disc with the line segment given by $\mathbf{P}' - \mathbf{P}_{\text{Old}}$. This method is described by the pseudo code of **Fig. 5**.

This mode of collision detection considers the racket normal. Consequently, the player has to control also the racket orientation during a game. To simplify the game, a player can switch off the racket orientation sensibility, and choose the racket normal to be normalized to the racket velocity. When the racket is not moving, the current racket normal has to be taken. This mode is easier for the player as he may neglect the racket's orientation. Therefore, he can concentrate completely on the ball and his stroke.

3.1 Response after ball - racket collision detection

In order to compensate certain network induced problems and to allow multiple game levels, we propose several methods for the ball-racket collision response. These methods range from user friendly "missile like" balls flying automatically to the partner's racket, to physically based collision response with ball spin effects for advanced players on high performance networks and computers. In this section, we are presenting the following collision response methods:

- racket determined: determined only by racket velocity
- autonomous ball: missile like; it flies to a given goal
- mixed response: mix of the above methods
- physical response: approximation based on physical laws

Racket Determined. To facilitate the game, one possible response is to force the ball to fly into the direction of the racket velocity during the hit. Moreover, as the user defined racket movements can be very fast it is useful to clamp the initial velocity to a certain range. The speed clamping avoids unrealistic strokes, and allows the application to predefine certain game levels. In this method, immediately after the stroke, the initial position of the ball is placed slightly in front of the racket collision point according equation (7).

$$\mathbf{P} = \mathbf{Q} + \mathbf{V}_0 * dt \quad (7)$$

This measure helps to avoid multiple collisions with the racket. Multiple collisions can happen if the racket is accelerating during the stroke. Then, it can strike consecutively the ball several times. It may even happen that the ball bounces into the racket from behind when the racket is overtaking the ball. Therefore, after a ball-racket collision, we stop collision detection during the next two or three frames in order to avoid such multiple collisions.

Autonomous ball. The autonomous ball flies to the racket of the partner independently of the hit executed by the player such as illustrated in Fig. 6.

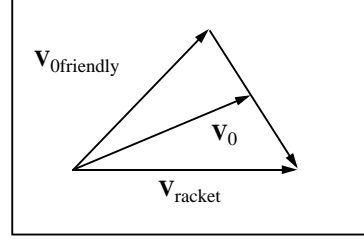
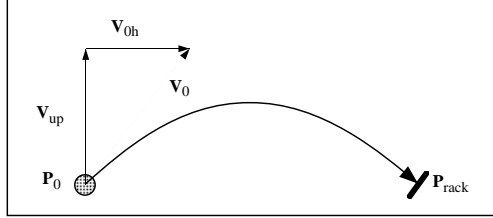


Fig. 6. “Friendly” ball flying to the partners racket **Fig. 7.** Mixed collision treatment

To simplify the calculation of the initial velocity of the autonomous ball, we suppose that the height of the goal point corresponds to the height of the ball position at the moment of the stroke. Given a vertical velocity V_{up} we can calculate the time t_0 the ball needs to mount and descend at the height immediately after the hit. Equation 8 shows the details.

$$\begin{aligned} P_{0y} &= P_{racky} \\ V_{up} \cdot t - 0.5 \cdot g \cdot t^2 &= 0 \\ t_0 &= V_{up} \cdot 2 / g \end{aligned} \quad (8)$$

In consequence, this time t_0 determines the horizontal velocity V_0 according to equation 9.

The vertical speed V_{up} can be chosen freely, or it can be matched to the actual vertical speed of the racket in order to have some limited influence of the player on the resulting ball trajectory. Finally, we can multiply the ball velocity V_0 by a factor that we call *friendlyFactor*. When this factor is 1, the ball flies automatically to planned point. If it is smaller than 1, it will bounce first in front of the planned point. If it is bigger than 1, it will fly over the given goal. Thus, the degree of “friendliness” of the ball can be adjusted easily according the taste of the players.

$$\begin{aligned} P_{0y} &= P_{racky}, \quad d_1 = P_{0x} - P_{rackx}, \quad d_2 = P_{0z} - P_{rackz}, \quad dist_h = \sqrt{d_1^2 + d_2^2} \\ V_{0y} &= \frac{dist_h}{b(1 - e^{-t_0/b})}, \\ f &= \frac{V_0}{dist_h} = \frac{1}{b(1 - e^{-t_0/b})} \\ \Rightarrow \vec{V}_0 &= \begin{pmatrix} d_1 \cdot f \\ V_{up} \\ d_2 \cdot f \end{pmatrix} \cdot friendlyFactor \end{aligned} \quad (9)$$

Mixed treatment. On the long run, the response of the autonomous ball is boring for advanced players using fast networks and powerful computers, because the ball always flies to the opponent, nearly independently of the actual stroke of the player. To improve the response behavior we can make a mixed collision treatment by taking into account the racket velocity.

$$\vec{V}_0 = \vec{V}_{0\text{friendly}} + (\vec{V}_{\text{racket}} - \vec{V}_{0\text{friendly}}) \cdot (1 - \text{friendlyFactor}), \text{friendlyFactor} \in [0..1] \quad (10)$$

As indicated in **Fig. 7**, we can calculate the difference vector between the autonomous ball velocity and the racket velocity. According to equation (we can adjust the behavior of the response with the "friendlyFactor", having a value between zero and one. When the friendly factor is 1, then the ball flies directly to the partner's racket. If the friendly Factor is 0, the racket speed will completely determine the ball's response. With values between zero and one we can choose a mixed behavior between the two extremes.

Physical Response. For advanced players using fast networks and powerful computers, a physical collision response model is convenient. The model we are presenting now is not an exact physical model, but it is based on physical collision principles in order to simulate real ball-racket collision responses. We consider also possible spin effects, which are produced if the racket normal and the racket speed at the collision are not parallel. We make the following assumptions:

- The new spin of the ball is determined only by the racket velocity component which is orthogonal to the racket normal.
- The spin response on the racket is proportional to the difference of the ball spin and the new spin induced by the racket movement
- In order to be able to hit with both racket sides, the racket normal is always in direction of the ancient ball position if a collision is detected. If a collision is detected, the actual ball position and the ancient one are at different sides of the racket. This means that the racket plane normal has to be inverted if $dP_{\text{old}} * n_{\text{racket}} < 0$ (see **Fig. 9**).

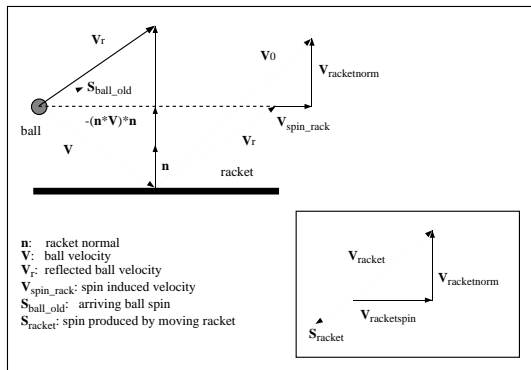


Fig. 8. Mixed collision treatment

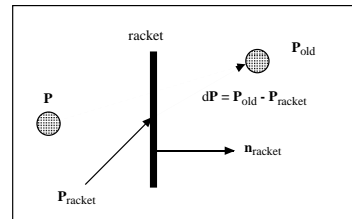


Fig. 9. Racket normal inversion

The response calculation is illustrated by Fig. 8. The velocity \mathbf{V} , the spin \mathbf{S}_{ball_old} , the racket normal \mathbf{n} , and the racket velocity \mathbf{V}_{racket} are given values of the arriving ball. From these values we deduce according to equation 11 the resulting velocity \mathbf{V}_0 after the hit which is influenced by a bounce component (or reflected velocity) \mathbf{V}_r , by the racket velocity $\mathbf{V}_{racketnorm}$ in direction of the racket normal and a spin induced component \mathbf{V}_{spin_rack} . The ball spin after the collision and its response at a bounce are given by equation 12 and 13. Additionally, after a bounce the ball spin is also damped by a given factor according to equation 14.

$$\begin{aligned}\vec{V}_0 &= \vec{V}_r + \vec{V}_{racketnorm} + \vec{V}_{spin_rack} \\ \vec{V}_r &= \vec{V} - 2 \cdot (\vec{n} \cdot \vec{V}) \cdot \vec{n} \\ \vec{V}_{racketnorm} &= (\vec{n} \cdot \vec{V}_{racket}) \cdot \vec{n} \\ \vec{S}_{racket} &= \vec{V}_{racket} \times \vec{n} \\ \vec{V}_{spin_rack} &= (\text{spinAirDamping} \cdot \vec{S}_{ball_old} - \vec{S}_{racket}) \times \vec{n}\end{aligned}\tag{11}$$

$$\vec{S}_{ball} = \text{spinRackDamping} \cdot \vec{S}_{racket}\tag{12}$$

$$\vec{V}_{spin_floor} = \text{spinFloorDamping} (\text{spinAirDamping} \cdot \vec{S}_{ball}) \times \vec{n}\tag{13}$$

$$\vec{S}_{ball} = \text{spinFloorDamping} \cdot \vec{S}_{ball}\tag{14}$$

4 Parameterization

In networked shared environments with asynchronous process communication we typically use a ball server managing dynamics and collisions of the ball. It communicates with game clients delivering positions of objects susceptible to hit the ball. Its task is to broadcast the updated ball and shared object data. In such a configuration we always have to deal with update delays of object positions and orientations which can range from milliseconds to one or several seconds in extreme cases. When the objects in the virtual world are moving slowly, the effects of the network delays do not disturb the animation. If, however, the objects are moving fast, such as a tennis ball, or a racket hitting a ball, then these delays can become disturbing, because each participant client displays temporarily a different object configuration. Especially, when collision detection between the fast moving ball and rackets has to be done, some consistency problems arise. **Fig. 10** shows a typical example of a ball-racket collision detected in the environment of the ball server. In the player client, however, the racket passes in front of the ball without touching it. Despite of this fact, the ball reacts as if being hit by the racket. This effect results from the fact that the ball movement is calculated in the ball server and the racket movement is captured in the player client. Both clients broadcast their object positions through the network. In our example the ball server gets the racket position with a

delay of one frame. Similarly, the racket client gets the ball position also with a delay of one frame.

Update losses represent another type of problems. If racket clients send asynchronously a lot of update information to the server, they can overload it. Then, it cannot broadcast each update in time. The next time, when it finally broadcasts the update information, it will take only the last update for a given object. Consequently, the more recent updates are ignored. This effect can lead to discontinuous ball and racket movements making collision detection and response difficult.

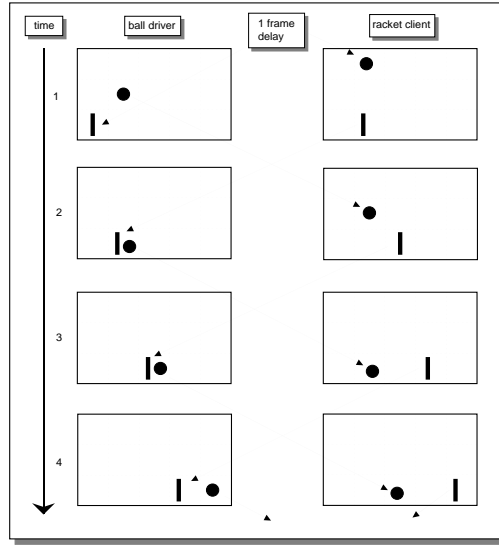


Fig. 10. Network delay

To minimize these delay problems we propose the following measures:

- Adjustment of ball speed and update rate to the network performance
- Introduction of game levels
- Extrapolation of racket movement

In networked applications it makes sense to include an absolute frame rate control which allows us to adjust the frame rate to the network performance. We measure the time a single frame takes for being calculated and force the process to sleep the rest of the given frame time interval. Independently of this real time frame rate control, we can adjust the time base of the animation, which allows us to adjust the ball speed.

The second step for minimizing network delays is the extrapolation of the racket positions in the ball server. An extrapolation can only work well for correlated movements. When the player makes fast, random or uncorrelated movements, then a prediction is impossible, and the resulting visual effects are worse than without an extrapolation. When the racket, however, executes a hit, generally the movement at the moment of the collision is correlated and extrapolation is possible. We use only linear extrapolation by adding an estimated offset to the actual racket position as given by equation 15.

$$\vec{P}_{extrapolation} = \vec{P}_{current} + \vec{V}_{current} \cdot extrapolationTime \quad (15)$$

The third possibility to deal with network problems is the introduction of game levels and simplifying approximations for the racket-ball collision detection and response. These measures allow inexperienced players to play with a certain success rate even at a low network performance.

The described models of the ball animation contain many parameters which can be used to define game levels, or/and to adjust a game to actual network and computer performances. The following list describes parameters acting on the ball dynamics, fixing its actual behavior.

- mass of the ball
- time step of the ball animation
- velocity and spin damping at collisions
- air friction
- the racket size for collision detection

Other parameters are useful to adjust the game to network performances and game-levels.

- Collision detection sleep rate (number of frames where no collision detection is done immediately after a collision (avoids multiple collisions))
- threshold values for collision sound and floor-ball collision detection
- frame time
- racket speed clamp values
- ball-racket collision detection mode (sphere, disc)
- ball collision response type (friendly, mixed, physical)

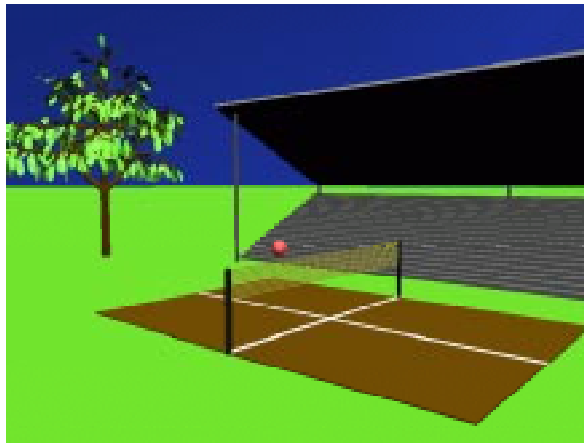


Fig. 11. A Rewriting system based VR ball game environment

For game applications these parameters should figure in configuration files and/or in interactive menus. For an easy game level on a slow network, we can use the distance dependent collision detection with an autonomous ball response and a friendlyFactor of one (see equation 10). A mid game level configuration corresponds to a collision detection with the racket where the normal is defined by the velocity, as well as a mixed collision response with a friendly factor of 0.8. Finally, an advanced game level on a ATM network, for example, with fast computers at each end could use a geometrical collision detection between the ball and a racket where the racket

normal corresponds to the real racket normal and a spin dependent physical collision response.

Parts of the generic ball model were successfully used in a networked VR tennis game described in [4]. At present it is implemented in a rewriting system based VR system [7, 8] at the Multimedia Laboratory of the University of Zurich, and it serves as a test-bed for networked VR ball games (see Fig. 11).

5 Conclusions

One of the difficulties of networked ball game models is the presence of fast moving objects. Whereas slow moving objects can be animated without problems, the fast moving ball and rackets require the use of special techniques, as we have to deal with low update rates and information losses.

For the ball animation we use simplified collision detection mechanisms as well as analytical movement and collision response calculation. These different techniques can be used to introduce game levels as well as to compensate network and performance problems.

Acknowledgments

Important parts of the research were realized at the Computer Graphics Laboratory (LIG), directed by Prof. D. Thalmann at EPFL, in close collaboration with MIRALAB, directed by Prof. N. Magnenat-Thalmann, at the University of Geneva. The authors would like to thank the directors and all members of both labs for their support and collaboration.

References

1. H. Noser, D. Thalmann, Sensor Based Synthetic Actors in a Tennis Game Simulation, Proceedings Computer Graphics International 1997, Hasselt, Belgium, June 24-28, 1997, pp 189-198
2. 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
3. D. Thalmann, H. Noser, Z. Huang, Chapter: *How to Create a Virtual Life ?*, in *Interactive Computer Animation*, eds. N.M. Thalmann, D. Thalmann, Prentice Hall Europe, 1996, pp 263 – 291
4. T. Molet, A. Aubel, H. Noser, T. Capin, E. Lee, I. Pandzic, D. Thalmann, N.M. Thalmann, Sannier, *Anyone for Tennis?*, Presence, Vol. 8, No. 2, April 1999, 140-156
5. R. Bronson, *Modern Introductory Differential Equations*, Schaum's Outline Series, McGRAW-HILL, Inc, 1973, Chapter 9, and 17
6. Priamos Georgiades, Signed Distance from Point to Plane, in *Graphics Gems III*, Ed. D. Kirk, Academic Press, Inc. 1992, pp. 223-224

7. 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
8. H. Noser, D. Thalmann, *A Rule-Based Interactive Behavioral Animation System for Humanoids*, IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 4, October-December 1999