Broadcast Games and Digital Television

Simon Gibbs, Michael Hoch, Hubert Le Van Gong, Richter A. Rafey, Sidney Wang
Sony Electronics Distributed Systems Laboratory
San Jose, California
{simon,micha,lvg,rafey,swang}@arch.sel.sony.com

Abstract

Digital television (DTV) allows simultaneous transmission of data along with traditional AV content. It provides an inexpensive and high bandwidth data pipe that enables new forms of interactive television and also new types of games, and other applications, for game consoles. This paper explores the potential role of DTV in future game titles and addresses how to merge television viewing with game play.

The paper introduces *broadcast games*, a new form of viewer experience, enabled by DTV, that combines elements from both television and gaming by using real-time data within the DTV signal to update game state. As an example, we focus on an auto-racing scenario and describe a prototype we have developed in this new genre. Auto racing has always been a sport where many leading edge technologies are tested and is a good candidate for combining AV content, real-time data and a 3D game engine. Features such as CG visualization of races and racing live against the real drivers are described.

1. Introduction

Digital television broadcasts now reach tens of millions of receivers worldwide. In Europe, Asia and the US, digital satellite television and digital cable television have been available for several years and have a growing viewer base. In the U.S., the Federal Communications Commission has mandated a transition period from analog NTSC over-the-air broadcast to its digital successor, ATSC, by the year 2006.

The current generation of DTV receivers, primarily cable and satellite set-top boxes (STB), generally offer limited resources to applications. From a manufacturer's perspective, the goal has been building low-cost receivers comprised of dedicated hardware for handling the incoming MPEG-2 transport stream: tuning and demodulating the broadcast signal, demultiplexing and possibly decrypting (e.g., for pay-per-view) the transport stream, and decoding the AV elementary streams. The focus has been on the STB as an AV receiver rather than as a general-purpose platform for downloaded applications and services. However the next generation of DTV receivers will be more flexible for application development. Receivers are becoming more powerful through the use of faster processors, larger memory, 3D graphics hardware and disk storage.

Within the software found on a DTV receiver, often called middleware, one of the main components is the presentation engine. The presentation engine performs rendering and compositing for applications running on the receiver. Current presentation engines use 2D graphics and a compositing model based on overlay planes. While this is sufficient for applications with simple graphics requirements, such as "electronic program guides", it is far from what is needed for an immersive game experience as provided by modern 3D game engines. However as 3D hardware starts to appear on DTV receivers, presentation engines are likely to become more game-engine like in functionality. An important question then for the gaming industry is to determine how games can take advantage of both the expected convergence between game engines and presentation engines, and the additional capabilities of DTV receivers.

This paper considers the above question and looks at new forms of gaming that make use of an integrated presentation engine/game engine. We introduce the notion of *broadcast games*, applications that combine

gaming with live AV content. We also describe a prototype auto race system we have implemented and plan to test over ATSC broadcast.

2. Broadcast Games

Digital television broadcast services, whether satellite, cable, or terrestrial, are based on the MPEG-2 standard. In addition to specifying audio/video encoding, MPEG-2 defines a transport stream format consisting of a multiplex of elementary streams. The elementary streams can contain compressed audio or video content, "program specific information" describing the structure of the transport stream, and arbitrary data. Standards such as DSM-CC [5] and the more recent ATSC data broadcast standard [1] give ways of placing IP datagrams in elementary data streams.

There are several ways the data delivery capability of DTV can be used to enhance the gaming experience. Perhaps the simplest is as a high-capacity distribution mechanism for game updates. For example, new levels, characters, etc. can be periodically broadcast and then cached by receivers. Already in the U.S., companies that aggregate unused ATSC bandwidth are well positioned to provide such a service. Furthermore, the subscription model – back-end billing database and "conditional access" (content encryption and key distribution) – used by satellite and cable operators for premium channels and pay-per-view, offers a revenue channel for the provider of game updates.

Our interest here, however, is more in situations where the AV portion of the DTV signal is directly tied to the game – what we call *broadcast games*. Broadcast games are game-like applications that accompany television programming or rely on data provided by the broadcaster. The natural areas for broadcast games are those forms of television that already have an affinity to game play, two are readily apparent – TV game shows and sports. In this paper we will focus on sports events. The basic concept behind a broadcast sports game is that information about the actual event is made available to a game engine as the event progresses. This real-time data enables several new forms of game play. The type of game play will vary from sport to sport, but in general ranges from simple question/answer scenarios to situations where the game player competes against the live competitors.

There are several reasons why sports is compelling area for broadcast games. First, a wealth of data (e.g., standings and statistics) is already produced during many sports events and potentially available for broadcast. Second, specialized tracking systems for sporting events are starting to be used by broadcasters. These tracking systems capture camera movement and movement of "objects of interest" such as balls, players, cars, etc. Sending this data via a reliable data channel to the game engine allows us to "synchronize" the game with the live event. And third, there is already a well-established gaming sector dealing with sports titles and there is much experience with sports-related modeling and simulation.

3. Related Work

Our work on broadcast sports games builds on two areas: attempts at offering a "play along" element for television or sports, and systems that provide real-time data for sports events by tracking object and camera movements.

Several television game shows are experimenting with formats allowing viewer participation. Columbia TriStar has produced interactive versions of *Wheel of Fortune* and *Jeopardy* that are preloaded with ATVEF [6] broadcast triggers to enable viewers to play along with the game while it is aired. The interfaces are adapted to TV viewers and run on the WebTV platform. The interaction takes place on the client, but scores may be uploaded and compared to other viewers to enhance the social competitive element. As another example. ABC offers a play along form of *Who Wants to be a Millionaire* through the use of a companion web site.

A few game companies have made a first step in the area of live gaming by offering the possibility of racing against the drivers. One notable example, AniVision's *Net Race Live* for the IRL (Indy Car Racing League), allows several thousands of people to play online against the real drivers. In Europe Kalisto has a similar feature

for FIA GT Championship racing based on its game called *Ultimate Race Pro*. Live race games for the two most popular leagues – Formula One and NASCAR – are not yet available but next year Sportvision will start offering real-time car position data from NASCAR races.

Several companies have developed specialized tracking systems for collecting real-time data from different types of sports. These tracking systems are now used by broadcasters to add graphics elements that are "registered" or "locked on" to the video. An early example is the Sportvision's FoxTrack hockey puck tracking system [4]. FoxTrack allows the broadcaster to render glows or streaks where the hockey puck appears in the video frame. More recently Sportvision has developed a system for rendering a virtual "1st and 10" line now used in many NFL broadcasts. Another form of registered graphics enhancement, also targeted towards sports broadcasts, is the insertion of images (typically advertising logos) registered to physical surfaces at the event site (e.g., the playing field, existing billboards). Orad and Princeton Video Image are among the pioneers of this technique. As a recent example, in NBC broadcasts from the summer Olympics in Sydney, shots of the swimming events had flags inserted into the swimming lanes to help viewers identify the competitors.

Extremely accurate camera tracking is the basis for inserting registered graphics in live video. Camera tracking equipment is well known from virtual studios [7] and special-effects work, however live production with high-zoom cameras, as is common for sports, adds to the challenge of collecting accurate camera data.

4. Example: Auto Race Scenarios

We now describe a set of broadcast game scenarios based on auto racing. We assume that the television signal contains coverage of a race event plus additional data that has been inserted by the broadcaster. This additional data is either generated by the broadcaster or obtained from third party sources (e.g., the race association or specialized data providers). The television signal is transmitted to a receiver where the data is extracted and passed to an application running on an integrated game engine/presentation engine. No back channel is assumed, but the following scenarios can be extended in several ways (e.g., multi-player, virtual leagues) if a back channel is present.

The data sent to the receiver is a combination of the data already produced (and used by broadcasters) and tracking data that requires additional equipment. It consists of:

- *standings data* – Standings data is typically compiled by some entity designated by the racing authority. It will typically include the actual standings plus derived information (e.g., number of laps led, best lap times). Standings data is already made available to broadcasters.
- *telemetry data* – The cars used in Formula One, NASCAR and other race leagues are often equipped with sensor devices that measure and transmit operational data (e.g., car velocity, gear position, etc.). Currently, this "telemetry" data is used by pit crews to monitor car and driver performance and by broadcasters to generate on-air graphics.
- *car tracking data* – Car tracking data gives the position (e.g., in GPS coordinates) of the cars in the race. This data is difficult to obtain in real-time and, depending on the tracking technology used, may have an error of a meter or more.
- *camera tracking data* – Camera tracking gives the position and orientation of broadcast cameras plus their zoom and focus settings.

The auto racing scenarios that we have developed offer different levels of interactivity. The broadcast game should complement, not compete against, the television programming. We recognize that in many cases viewers will just want to watch the broadcast and not be distracted by a game. On the other extreme are those who want a racing game allowing them to play along and actively participate in the live event; for such viewers, the main benefit of the DTV feed is to get more out of their game. It should also be recognized that there is a wide middle ground between these two extremes. Here interaction may be sporadic and left to the discretion of the viewer – for instance, they might choose to answer a trivia question, or play a simple game during a lull in the TV action.

Fantasy

Fantasy teams are commonly known from football, and recently fantasy team play for other sports like baseball, basketball, hockey, golf and auto racing have become popular on the Internet. In fantasy leagues, points are collected for a user defined team or collection of drivers. These points are usually updated after an event, but can be updated in real time if standings data is added to the video broadcast of the event.

With the data streams described above, new forms of play along scenarios are made possible and the real-time updates of fantasy scores can be displayed while watching the broadcast. In car racing, for example, the user could select a team of drivers and collect the points each driver gets in the race. Additionally, any driver who leads at least one lap during the race receives some bonus points, or the driver who leads the most laps during the race receives some extra points. With a live broadcast this scheme can also be extended to gaining points for passing. User interaction can be incorporated by predicting the time for a pit-stop, or answering trivia questions that accumulate to the fantasy points. These questions can also be tied to the sponsors of the race and create potential revenue models.

Assuming the broadcast stream contains the standings information of the cars in the race and camera information of the current video stream, further interactive features are possible that enhance the play along experience. One obvious one is the display of the fantasy standings data, i.e., the standings customized to the user's selected fantasy team (see Figure 1a) or the real-time presentation of trivia questions (Figure 1b). With the help of the camera data, and car position data, it is possible to highlight a selected car in the video feed, e.g., draw a halo around it. Hence, the user is able to highlight his favorite driver when he comes into view, or go through the list of his team's drivers and have them highlighted in the video feed.

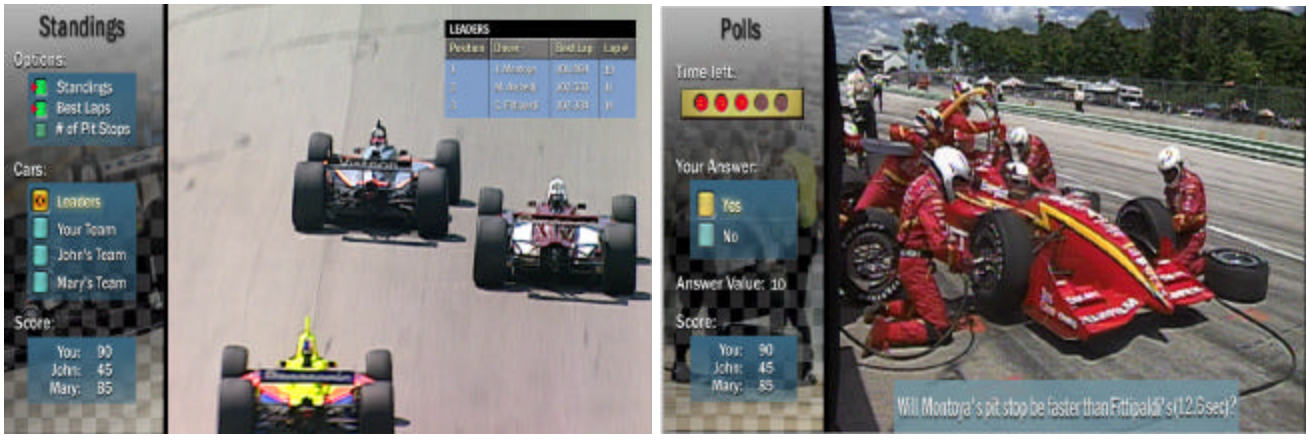

Figure 1: a) Fantasy Score, b) Polls and Trivia Questions

With 3D rendering capabilities on the receiver site, we can further enhance the viewing experience by allowing a "virtual" mode. By pressing a button on the remote, the user will fade from the video feed to a virtual scene that is frame-aligned with the video feed. The viewer can continue watching the race in virtual mode or switch back to video. Virtual mode is made possible by using registered cameras at the event site and interpreting the camera and car tracking data on the receiver site in order to render a live visualization of the event. The virtual mode allows a viewer to follow a favorite driver from uncommon camera perspectives, e.g. helicopter close-up view, or to follow the fantasy team even if the drivers are currently not leading (the broadcast coverage is often focused on leading drivers). This CG-viewing capability allows a close-up focus on the fantasy team while not loosing track of the race itself. Since the location and registration of the event cameras is known, one can allow the user to select a camera in the virtual scene that represents a real camera, hence maintaining the original camera work. We believe it will also open up other possibilities of gaming that are connected to a live event.

Live Game

The *live game* scenario falls on the other end of the spectrum of gaming activities tied to a DTV broadcast. It is intended for viewers who wish to actively interact with the broadcast by racing against the real drivers.

The car simulators used by current game engines have attained the level of accuracy where drivers get similar times on the simulators and real tracks. By running a simulator on a receiver and connecting it to the telemetry and tracking feeds from the DTV, the game engine can now replace its AI simulation by updates received from its data feed. This adds even more excitement to playing these games, as players will be able to compare themselves to the real professional drivers. Viewers could also become members of their fantasy teams and earn additional points (e.g., by passing real drivers). If IP connectivity is available, players from other homes can also be added to the fantasy teams.

5. Test Implementation

We are currently developing a test system that uses ATSC broadcast for delivery of an auto-race broadcast game. The ATSC data pipe is about 19.4 Mbps. We are working with SDTV (standard definition television) AV content, rather than HDTV. SDTV requires about 5 Mbps for sports video. Although over two thirds of the ATSC pipe is then unused and potentially available for data delivery, broadcasters will likely fill most of the unused bandwidth with additional SDTV signals or other data services. Consequently, our system has a more conservative data budget of 200 Kbps. This is also within the limit of what can be provided by current broadband connections (xDSL or cable modem) and so leaves open the possibility of separated delivery of the data and video feeds.

Our test system supports a variety of interactive scenarios including fantasy teams and polling as described in the previous section. We have also modified a game engine to use real-time feeds to update game state. Currently we are working with a data set assembled after a race and not providing full coverage of the event. By the end of the year we expect more complete data to be available and are aiming for an end-to-end transmission test in 2001.

Test System Configuration

The features found in our prototype are intended for a DTV receiver with an integrated presentation engine/game engine environment. At the time of development we did not have such a platform available, so our test system uses a “two box” receiver shown in Figure 2. Although the presentation and game engines run on distinct processors, they are coupled by a network link allowing the application to control both engines.

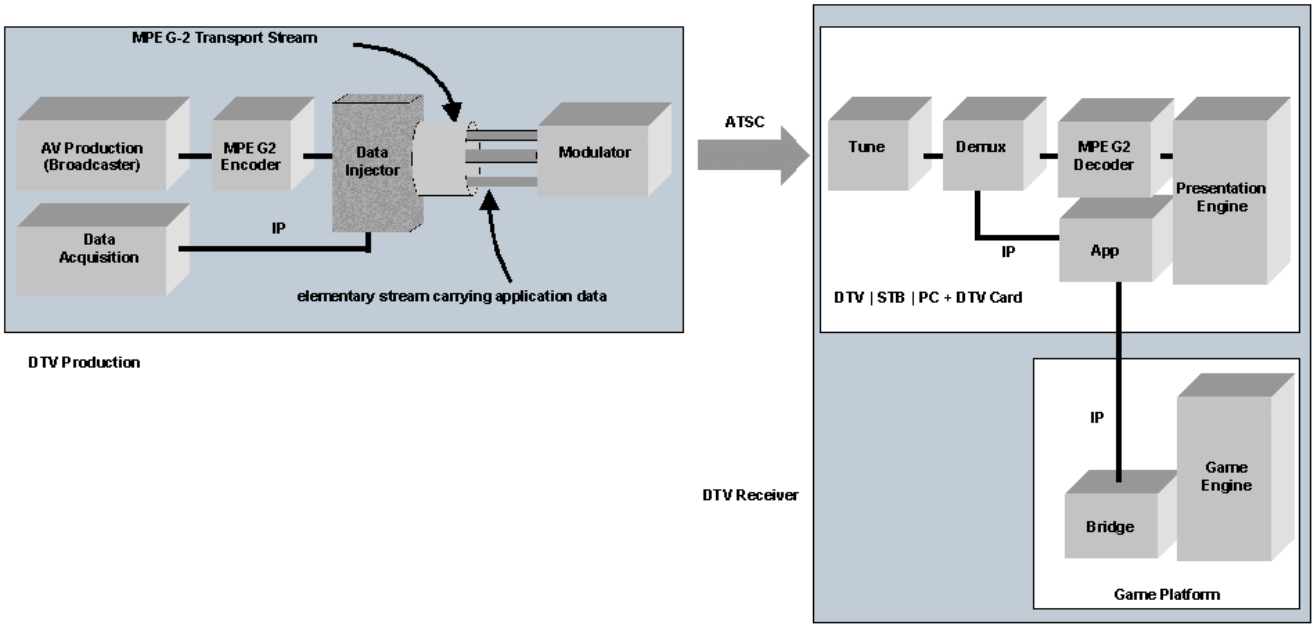

Figure 2 Configuration of end-to-end test system

Figure 2 illustrates the main hardware and software components in our end-to-end test system for broadcast games. The left-hand side depicts the production of an MPEG-2 transport stream with encapsulated IP data. The configuration shown here differs from the typical DTV production chain through the introduction of the two components labeled *Data Acquisition* and *Data Injector*. The data acquisition component handles the various real-time data sources made available to the broadcaster. For our car racing example application, this component obtains the camera tracking, car tracking, car telemetry and standings data feeds and converts these into IP packets which are then sent to the data injector. The data injector receives the IP packets and encapsulates them in an elementary stream that is multiplexed with the AV elementary streams. The resulting transport stream is then modulated and transmitted to the receiver device.

The DTV receiver tunes to a DTV signal, demodulates and demultiplexes the incoming transport stream, decodes the AV elementary streams, and passes the result to the presentation engine. The DTV receiver also extracts encapsulated IP data from elementary streams and passes this to the application. The application is responsible for initialization and control of the presentation engine and game engine. It also relays incoming data to the game engine, and communicates with the game engine via the "bridge" component shown in Figure 2.

Presentation Engine Extensions

The presentation engine in our test implementation is called *Blendo* [2]. Developed by our colleagues at Sony, Blendo is derived from VRML but adapted to support the extensibility, performance, and programmability that are required to be suitable for interactive TV. Blendo exploits the graphics capabilities of emerging consumer platforms to provide *late composition*, i.e., composition on the receiver rather than prior to broadcast. Blendo enables consumer devices to deliver the production quality that traditionally could only be achieved with equipment in broadcast suites. The syntax and structure of Blendo is based on VRML, and as such these extensions are applicable to VRML in general.

To enable the features we wanted to showcase in our prototype, there are three major areas of extensions we built. These extensions are implemented as new Blendo nodes that allow declarative use and easy authoring in the VRML/Blendo syntax. The extensions are: registered graphics insertion, handling of streaming data, and live video integration.

Registered Graphics Insertion

Some broadcast game scenarios require the placement of graphic objects be correlated with real objects in the video. Since current camera and object tracking systems provide the data required for accurate graphics insertions registered with the video, we have developed new nodes in Blendo that support these data fields and allow a declarative representation for camera-aligned overlay graphics in the VRML/Blendo syntax.

Camera tracking equipment typically uses encoders to read the current pan, tilt, and twist of the camera, as well as, the zoom level, i.e., the field of view. Since the position of the real camera is tracked, the virtual camera can be placed so that it corresponds to the real camera. The next step is to render the graphics at the appropriate position and size using the virtual camera and, thereafter, composite the rendered set with the camera shot. We use a two-pass rendering technique that renders the scene that is to be used as a texture in the second pass. The virtual camera is implemented as a modified VRML Viewpoint node that adjusts itself in position, field of view, pan, tilt, and twist corresponding to the current video feed (the data is getting collected and routed using the ATSC_DataHandler node described in the next section). We also developed a parameterized GridNode as an extension to Blendo that renders a scene on to a grid. We then adjust the texture coordinates of the grid to correct for optical center shift and radial lens distortion of the real camera. The final texture is composited (overlaid) on the video feed. The lens correction becomes necessary if we want to insert graphics objects that are aligned with the content of the video feed. Without lens distortion correction, graphics objects can appear to slide over the video as the camera pans or zooms. We apply a correction technique that is related to the well-known techniques of rectification and geometric correction [8].

Streaming Data

While VRML provides an event model that enables triggering media events based on signals, there is no data architecture built into VRML beyond some simple field types. For this reason we have created a top-level extension node for handling the ATSC data stack called ATSC_DataHandler.

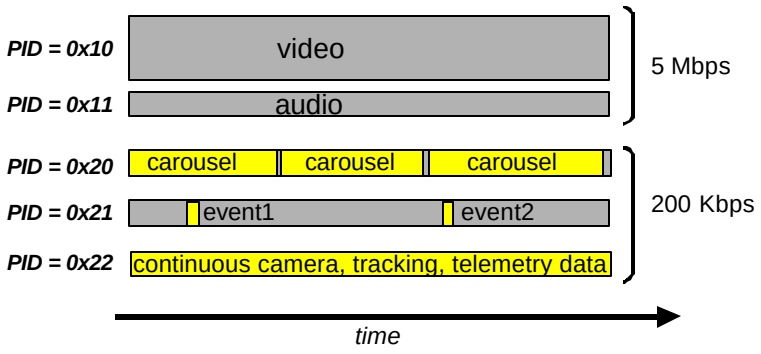

Figure 3 Transport stream used by test system.

As shown in Figure 3, referencing an elementary data stream is done through the MPEG-2 *program identification* field (*PID*) of the stream. A DTV receiver should have the ability to filter out any unwanted data streams and only process data streams with specific PIDs. Furthermore, we have defined and classified three types of data streams: *event stream, continuous stream,* and *carousel stream.* An event stream contains data that occurs sporadically. One example is the polling scenario where the broadcaster can insert trivia or polling questions anytime during the program. Typically this type of data will contain a MPEG-2 *presentation time stamp (PTS)* so that the compositor can use this information to present the data at the appropriate time during the broadcast. On the other hand, a continuous stream contains data that is updated throughout the entire program. Examples include camera tracking data and car position/telemetry data. For this type of stream, synchronization with the broadcast video is also via PTS values. Finally, we have defined a type of stream called carousel. Data contained in the carousel are looped repeatedly during the broadcast. For example, in motorsports, statistical data (e.g. current standings, current lap, etc.) can be carouseled so viewers who tune in during the middle of the

broadcast can access this information at the next carousel cycle. Also, user interface assets and configuration data that are specific to a particular race can be placed in the carousel.

One feature we are exploring is provisioning applications that persist over multiple broadcasts, e.g., an interactive application for the entire racing season. Assets that remain the same over time are stored locally on the presentation platform, while assets specific to particular races (e.g., track model, sponsor logos) are downloaded via the data carousel.

Live Video Integration

VRML97 supports a MovieTexture primitive to present video clips, but streamed video from a broadcast is not directly supported. Blendo introduces a new level of abstraction to support video synthesis, called *surfaces*. By using this abstraction, Blendo's architecture enables arbitrary content (e.g., video, HTML, Flash) to be rendered into the scene at the appropriate frame rate without burdening other elements (e.g., a 5 fps animation on one surface would not prevent video on another surface from playing at 30 fps). Blendo introduced a MovieSurface node to capture the semantics of controlling and displaying full frame-rate video. In our work, we have subclassed the MovieSurface to support a live video stream (vs. locally stored video) as a VideoSurface node that takes decoded video from a Hauppauge ATSC receiver/tuner card [3] and displays it on the surface.

Game Engine Extensions

On the pure gaming side, we modified a car-racing simulator so that it could receive remote car information such as speed, position and orientation on the track instead of using its AI engine to simulate the cars. This is similar to a networked version of such a game, except that the information provided for the "real" cars did not perfectly match what the game engine needed. For instance, the position of a real car did not include the side-to-side position on the track. We then had to partly rely on the simulator to complete each car's state in the simulation.

One of the challenges when enabling gamers to race live against real drivers is maintaining consistency between the race game (and what the player experiences) and the real race. Since the live racers obviously don't see the virtual cars (from the game), they will not take any measures to avoid them. Hence, the collision detection algorithm used by the game needs some modifications. Several types of collisions can occur, they are represented by the four different cases in the following table:

bumps into	**real car**	**player's car**
real car	Case 1 (real world)	Case 2 (live game)
player's car	Case 3 (live game)	Case 4 (typical game)

Case 1 and Case 4 do not need any special treatment. Case 1 corresponds to a collision happening in the real race and so there is no decision to take about what is happening: the game just updates the cars' positions using tracking data. Case 4 is a typical game situation, there is nothing to change here. The two remaining cases involve changes to the game logic.

In Case 3, a player bumps into a real car. Whereas the game engine determines the effects on the player's car as if it were a normal collision, the behavior of the real car (in the game) depends on how the game handles tracking data. We have considered two possibilities representing different tradeoffs between realism and accuracy:

- The game uses *only* the received tracking data to position the real cars in the game. In this case, there is no possible effect on the rendered real car after a collision with a player. This solution is fairly straightforward from a design point of view, it is accurate regarding depiction of the real race but the lack of realism within the game may be disturbing to some players.

- The game may also use its simulator capabilities to compute a new position of the real car. In this situation, after a player collides with a real car, the game engine modifies the position/orientation of the (rendered) real car to reflect the collision. The game engine then progressively repositions the real car so that it matches the corresponding tracking data. Although technically more challenging, this represents a much more realistic behavior from the game player's perspective. The drawback may be an inconsistency between the live drivers' performance (lap time…) and their performance in the game.

The last type of collision, Case 2, is when a real car runs faster and bumps into a player's car. Once again, we identify two different ways of dealing with this issue:

- *Do not allow overlap between two cars*: A player's car is handled in the same way as in a classical race simulator and thus can be bounced out by an incoming real car. Though this might make the race difficult from the player's point of view, this represents the easiest way to implement a broadcast-driven game starting from an existing game.
- *Allow partial overlap between two cars*: Here, a real car might overlap a player's car without inducing any effect on the player's car on the condition that the latter was ahead at the time the collision occurred. Although this is more difficult to implement in a collision detection algorithm of a game, this would probably make the game more playable.

6. Future Directions

Broadcast games build a gaming experience on top of broadcast content. This paper has described some first steps in this direction in the area of auto racing. We plan to continue this work by studying how to better integrate live video content with a game experience. One challenging area is to realistically insert game elements, such as a game car, into the live video. This would allow viewers at home to see their friends or family members appear in the competition. This could be extended to a complete parallel competition with the best of the virtual players competing live against the pros and the virtual player being inserted in the live broadcast for all to see. Another example of video/game integration is what could be called a "texture channel" – using live video as dynamic textures in the gaming model. This would increase the visual richness of the game and allow it to more accurately reflect reality (e.g., weather conditions, shots of the stands, etc.).

References

[1] *ATSC Data Broadcast Standard (A/90)*, ATSC July 2000, www.atsc.org/Standards/A90/A90.html

[2] Broadwell, P., Kent, J., Marrin, C., and Myers, R. *Blendo: An Extensible Media Modeling Architecture*. http://www.web3d.org/fs_specifications.htm, 1999.

[3] Casey, J.B., Aupperle, K., Digital TV and the PC. http://www.hauppauge.com/html/dtv.pdf, November 1998.

[4] Cavallaro, R., *The FoxTrack Hockey Puck Tracking System.* IEEE Computer Graphics and Applications, 17(2):6-12, March-April 1997.

[5] *Enhanced Content Specification,* Advanced Television Enhancement Forum (ATVEF), http://www.atvef.com/library/spec1_1a.html, 1999.

[6] *Extensions for Digital Storage Media Command & Control,* International Standard ISO/IEC 13818-6, 1999.

[7] Gibbs, S. et al., *Virtual Studios: An Overview*. IEEE Multimedia, 5(1):18-35, January-March 1998.

[8] Niemann, H., *Pattern analysis and understanding.* 2nd ed. Springer, Berlin Heidelberg New York, 1990.