Enabling Custom Enhancements in Digital Sports Broadcasts

Richter A. Rafey Simon Gibbs Michael Hoch Hubert Le Van Gong Sidney Wang

Sony U.S. Research Laboratories, Distributed Systems Lab
3300 Zanker Road, San Jose, California 95134 USA
{rafey|simon|micha|lvg|swang}@arch.sel.sony.com

ABSTRACT

Digital TV is an area where interaction is expected to become increasingly prevalent in the next few years. Graphics technologies like VRML that are designed for distributing applications provide a presentation engine that can serve as a foundation for moving visual enhancements from the studio to the consumer platform. As these enhancements move downstream to viewers, they introduce capabilities for compelling interaction. We have developed a prototype that extends VRML to support interactive TV applications, specifically focusing on enhancements for sports broadcasts. Our extensions are built on Blendo [1], a presentation engine developed in Sony that derives from VRML. This paper describes the context for our extensions and the specific extensions that we have developed so far.

Keywords

Applications, Digital TV, Graphics Systems, Interactive TV, Multimedia, Sports Broadcasting, Video, VRML.

1. INTRODUCTION

Interactive TV is evolving as a compelling space for new media applications as platforms that enable interaction become available in more households. Digital TV (DTV) enhances the potential of interactive TV by providing a very high bandwidth data channel into homes that can be synchronized with A/V broadcasts. The Broadcast Virtual Worlds (BVW) project in Sony's U.S. Research Laboratories is exploring the features of new applications that can emerge to differentiate DTV from traditional analog broadcasting.

Sports broadcasts represent a particularly interesting domain for interactive TV. Sports are inherently competitive and thus lend themselves to play-along scenarios. There is also a tremendous amount of associated data that can be challenging to present effectively for a mass audience. The nature of most sports programming being live is unique in that producers must be able to dynamically insert interactive enhancements to complement the action of the event. New innovations in camera tracking are addressing sports as an early target domain for graphical enhancements that are visually registered with the footage (e.g., the orange first down line in NFL football broadcasts).

Our research builds in part on the premise that moving control of broadcast enhancements from the studio (for mass presentation) into the living room (for personalized consumption) is one way of enabling interaction. Our colleagues at Sony have developed Blendo [1], a presentation engine 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*. It enables consumer devices to deliver the production quality that traditionally could only be achieved with equipment in broadcast suites. The BVW project uses Blendo as the presentation engine in our development. The syntax and structure of Blendo is based on VRML, and as such these extensions are applicable to VRML in general.

We have developed four classes of extensions to make Blendo suitable for sports enhancements that result in new Blendo/VRML nodes:

- Live video integration
- Registered graphics insertion
- Streaming data
- Interaction via remote control

These extensions enable a graphics-capable platform (e.g., a PC, set top box, or game console) to interpret Blendo/VRML content and render the graphics under user control.

In this paper we describe how we applied Blendo and these extensions to develop a prototype to demonstrate interactive scenarios in an auto-racing broadcast. Section 2 describes current work in digital broadcasting and sports enhancements. Section 3 describes the prototype we have developed to validate and refine our ideas. Section 4 addresses the specific extensions we built to support key features through the declarative Blendo language. We conclude with our results and research directions.

2. BACKGROUND

Several companies have developed systems for inserting registered graphics into live video. These systems are typically targeted towards sports broadcasts and use a combination of camera tracking and object tracking technologies. An early example is the FoxTrack system from Sportvision [2]. Using positional information obtained from IR transmitters in hockey pucks, visual enhancements such as glows or streaks can be rendered 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.

A second 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 [3] is one of the pioneers of this technique, which again uses camera tracking and chroma-keying techniques. 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 as a way of helping viewers identify the competitors.

Our work uses similar techniques for rendering and compositing, but differs by moving the processing from studio tools to a graphics-capable client in the home. Emerging digital media standards like ATSC [4] provide the means of transmitting the necessary data streams to consumer platforms. VRML97 [5] provides a foundation for dynamically updated graphics content on distributed platforms, but it lacks several features that are critical to support these kinds of visual enhancements. Blendo provides an extensible version of VRML where we can fill in these gaps to move sports enhancements downstream to consumer platforms.

3. PROTOTYPE: AUTO RACING

For our prototype, we have selected auto racing as the target domain. Auto racing is a compelling starting point for exploring interactive TV for a variety of reasons. First, *live streamed data is collected automatically*, which reduces the requirement for authoring while still providing a data-rich environment. The fact that there is *a large international audience* enhances the potential reach of our work and grounds it in a familiar domain. Since we are focusing largely on the role of computer graphics in our applications, auto racing simplifies the process in some ways because of the fairly *constrained visual representation* (mostly rigid cars circling a fixed track).

3.1 Prototype Features

Our prototype covers a set of experimental interactive scenarios. For the purposes of this paper we focus on three of these.

Polling

A relatively simple interactive feature presents questions that test viewers' knowledge or ask them to make predictions. By introducing scoring that endures (and potentially prizes), the interactive version appeals to the competitive nature of viewers and also introduces some social context by ranking the scores among viewers. The value of doing this as a local enhancement is that people can decide to play along and have the questions appear on the local presentation engine without cluttering the screen for those who are not interested in such features.

Cockpit

A good example of dynamic data that could be automatically delivered and presented based on local preferences is a cockpit view that shows a graphical representation of sensor data from the cars (speed, RPM, lateral force). There are systems in place to provide such a graphical representation as part of the broadcast stream, but broadcasters have to be very selective about when to present it to avoid annoying viewers. By moving this to a local presentation engine, we give viewers the flexibility to visualize sensors as they like and for whichever cars interest them. Since it is rendered locally based on data, it is possible to render alternative visualizations and "skins" based on viewer preferences.

Performance

Broadcast enhancements like the synthetic first down line are also strong candidates for customization, as they introduce a tradeoff between the visual purity of the event and the potential for optimally presenting interesting information. Purists may resent their presence, but others quickly adopt such features as key to the viewing experience. We explore this with a scenario where we showcase lateral acceleration mapped as a dynamic, color-coded arrow on the cars as they handle a particular turn. Rendering this locally requires camera-tracking data in addition to the car data (including position/orientation). The presentation engine must be able to process these various forms of data to present appropriate visual representations on demand.

3.2 Data Sources

Moving the processing from the studio to the home raises many issues for preparing data and populating a scene that are different from traditional VRML in the Web domain.

The real-time data available for interactive sports broadcasts will typically consist of data feeds obtained from several sources (e.g., the event organizing authority, specialized "data providers"). For a car racing application, possible data feeds include:

1) camera tracking data – camera extrinsic and intrinsic (Tsai model [6]) parameters.
2) car tracking data – car position and orientation data.
3) car telemetry data – car performance data (rpm, gear, etc.).
4) standings data – official standings data, typically calculated as cars pass particular points on the track

The data rate and synchronization requirements of the feeds will vary. For example, the camera tracking data is typically obtained on a per-frame basis and must be synchronized on the receiver with the corresponding video frame.

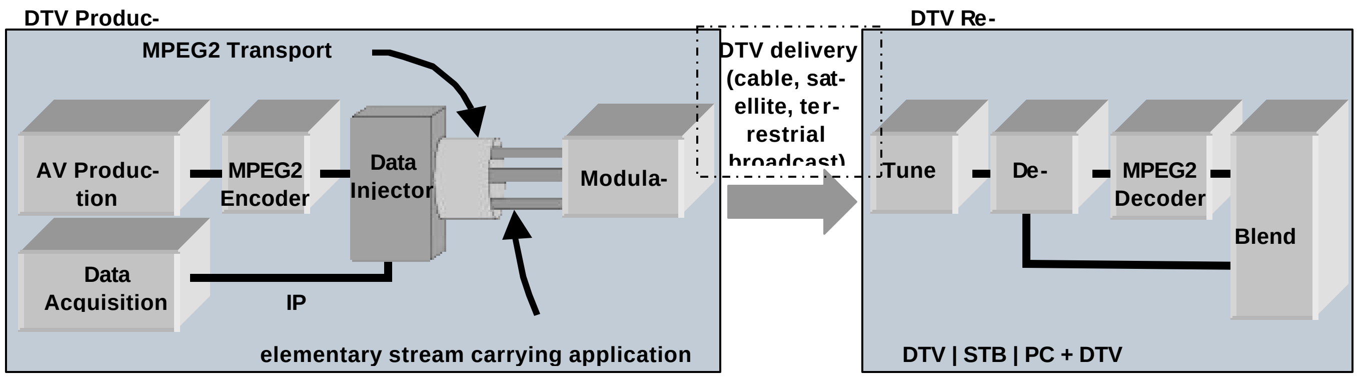


Figure 1: End-to-end system for delivering DTV content to Blendo

3.3 Testbed System

Figure 1 illustrates the end-to-end system that we are using for a testbed. The left-hand side depicts the production of an MPEG 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 D*ata Injector* in Figure 1. 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 A/V elementary streams. The resulting transport stream is then modulated and transmitted to the receiver device.

A DTV receiver is a device that tunes to a DTV signal, demodulates and demultiplexes the incoming transport stream, decodes the A/V elementary streams, and outputs the result. We call a DTV receiver “data capable” if it can in addition extract application data from elementary streams. The data capable DTV receiver is the target platform for our work. These devices can be realized in many ways: a digital set-top box that connects to a television monitor, an integrated receiver and display, or a PC with a DTV card. For our current development we are using the latter with the Blendo engine as the application.

4. VRML EXTENSIONS

To enable the features we wanted to showcase in our prototype, there are four major areas of extensions we built. These extensions are implemented as new Blendo nodes that allow declarative use and easy authoring. The extensions all have potential value in future Web applications as well, but they are essential for making VRML a suitable platform for interactive TV applications.

4.1 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 marking engines (e.g., video, HTML, Flash) to render into the scene at the appropriate frame rate without burdening other elements (e.g., a 5 frames/sec 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 30 fps video.

In our work, we have subclassed the MovieSurface to support a live video stream (vs. locally stored video) as a *VideoSurface* node. The semantic definition of the new node is shown as follows:

```
VideoSurface {
   field SFString videoSource "ATSC"
   field SFVec2f videoResolution 720 480
}
```

The *videoSource* field of the VideoSurface node indicates where the VRML browser is receiving video input. The possible values of the field are hardware dependent – for our particular testing platform there are three possibilities: *ATSC*, *COMPOSITE*, and *SDI*. In the case of *ATSC***,** the decoded video is extracted directly from a Hauppauge ATSC receiver/tuner card [7] and displayed onto the surface. In this configuration we are assuming that the Blendo browser and the DTV receiver reside in the same machine or set top box. Alternatively, one can envision a two-box setup, where the first box is a DTV receiver and the second box executes our Blendo compositor. The decoded video stream is sent from the DTV receiver to the compositor via either the *COMPOSITE* video port or the *SDI* (serial digital interface) video port.

The *videoResolution* field specifies the dimensions of the extracted video. Currently our compositor has the capability of handling full-sized NTSC video of 720x480 at 30 fps. The Hauppauge ATSC tuner card is able to down filter any of the ATSC video resolutions to 720x480.

4.2 Registered Graphics Insertion

Some classes of enhancements require placement that is correlated with objects in the video. Since current camera and object tracking systems provide the data required for accurate graphical insertions registered with the video, we have developed new nodes in Blendo that can support these data fields to allow a declarative representation for camera-aligned overlay graphics. The camera tracking equipment, well known from virtual studios [8], 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. Furthermore, the position of the camera is tracked in order to reproduce a virtual camera that 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. However, geometric correction that accounts for lens distortion and optical center shift is often not applied because of the increased processing cost. The correction becomes necessary if we want to insert graphical objects that are aligned with the content of the video feed. We apply a correction technique that is related to the well-known techniques of rectification and geometric correction [9], which are normally applied on a per-image basis. Here, we introduce a two-pass rendering technique that renders the scene that is to be used as a texture in the second pass. This texture is then corrected for distortion parameters (radial distortion and optical center shift) and finally composited with the current video image. Some current virtual set systems perform this correction since it becomes especially important if one has, for example, real objects sitting on virtual objects. Without lens distortion correction, real objects can appear to slide over the virtual set as the camera pans or zooms.

```
Gridnode {
   field SFVec2f dimension 10 10
   field SFVec3f distort 0 0 0
}
```

We developed a parameterized *Gridnode* as an extension to Blendo, e.g., 10x10 points with an IndexedFaceSet, that renders a scene with the virtual camera set so that it corresponds to the position, pan, tilt, twist, and field of view of the current video feed. The Gridnode's texture coordinates are adjusted to correct for the optical center shift and the radial lens distortion (general formula). The final texture will be composited (overlaid) on the video feed. Gridnode is derived from GeometryNode, and it allows the integration of camera tracking data in VRML syntax:

```
PROTO Gridnode
"urn:vrml:sony.com:native/nodes/Extensions#G
ridNode"
PROTO MyScene "scene.blo"
Transform {
   children [
      Shape {
         appearance Appearance {
            material Material { }
            texture Texture {
               surface SceneSurface {
                  children [
                     DEF scene MyScene {}
                  ]
               }
            }
         }
         geometry Gridnode {
            dimension 10 10
            distort FROM scene.distort
         }
      }
   }
}
```

Figure 2: Declarative use of Gridnode as a geometry

The scene (defined in the file *scene.blo*) that gets rendered onto the IndexedFaceSet of the Gridnode uses the *CameraViewpoint* node.

```
CameraViewpoint {
   <fields from Viewpoint omitted for clar-
ity>
   field SFVec3f distort 0.2 0.1 0.000194
}
```

CameraViewpoint extends the Viewpoint node such that it is able to accept camera data (the data is getting collected and routed to the CameraViewpoint node using the DataHandler node described in the next section) to adjust itself in position, field of view, etc., corresponding to the data. The *distort* parameter that is passed into the Gridnode holds the optical center shift in the x and y directions and the first order radial lens distortion parameter. We use a 3-dimensional vector here to simplify parameter passing.

```
field SFVec3f distort FROM Tracked-
Cam1.distort

PROTO CameraViewpoint
"urn:vrml:sony.com:native/nodes/Extensions#C
ameraViewpoint"
DEF TrackedCam1 CameraViewpoint {
   fieldOfView  0.635262
   distort 0.2 0.1 0.000194
   description   "MainView"
}
…
<other graphic objects omitted for clarity>
…
```

Figure 3: File *scene.blo* containing the graphic objects to be inserted with CameraViewpoint node that gets updated by tracking data

The field *distort* at the beginning of *scene.blo* uses a Blendo ROUTE statement (FROM) to access the *distort* field of TrackedCam1. This in turn allows this field to be accessed when *scene.blo* is instantiated. In this case the field is accessed to route the data to the Gridnode (Figure 2).

4.3 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. We developed a data architecture and have chosen MPEG-2 as our primary delivery mechanism to be in step with emerging digital broadcasting standards.

Current digital television broadcast services, whether satellite, cable, or over-the-air, 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 may be compressed audio or video data, information about the structure of the transport stream, and arbitrary data. Standards such as DSM-CC [10] and the more recent ATSC data broadcast standard give ways of placing IP datagrams in elementary data streams.

For this reason we have created a top-level extension node for handling the ATSC data stack called *ATSC_DataHandler* with the following interface:

```
ATSC_DataHandler {
   field MFInt32 pid  [0, 1, 2]
   field MFString  type ["INSTANTANEOUS",
                        "CONTINUOUS",
                         "CAROUSEL"]
   field MFBool active [TRUE, TRUE, TRUE]
}
```

Referencing an elementary data stream is done through the *program identification* field (*PID*). A DTV receiver should have the ability to filter out any unwanted data streams and only process those data streams indicated in the *pid* field with the *active* field set to *TRUE*. Furthermore, we have defined and classified three types of data streams: *instantaneous stream, continuous stream*, and *carouselling stream*. Instantaneous 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 *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 usually done via the timecode information. Finally, we have defined a type of stream called carousel. Data contained in the carousel are looped repeatly during the broadcast. For example, in motorsports, statistical data (e.g. current standings, current lap, etc.) can be carouseled so viewers who tuned in during middle of the broadcast can access this information at the next carousel cycle. Also, part of the graphical user interface (GUI) assets that are specific to a particular race can be placed in the carousel.

One feature we are exploring is enabling an enduring application that would persist over multiple broadcasts, e.g., an interactive application for the entire racing season. The assets that remain the same over time are locally stored on the presentation platform and assets specific to particular races are downloaded via the data carousel.

We have designed several packet types to format the data to be broadcast (e.g CameraTrackingPacket, CarTelemetryPacket...). These packets have been designed to reduce the bandwidth needs while providing high accuracy information. Once captured and gathered at the event site, data is sent over IP to the Data Injector that encapsulates the data into the MPEG-2 transport stream. We actually send one chunk of data per frame (29.97 fps) but this could easily be adapted to the needs of the application. On the receiver, the tuner board demultiplexes the transport stream and the IP packets are forwarded to the TCP/IP stack using a pre-defined Multicast group address (set in the Data Injector). The application listens to this Multicast group to receive the data.

4.4 User Interaction

Once the enhancements are under the control of the viewer, it is essential to make these accessible through an intuitive interface. Television is typically a very passive experience and consumer acceptance will fall off as the interface strays from the simple button press on a remote control. Whereas Web-based content typically involves a mouse-driven cursor that can point to an arbitrary region of the screen and thus VRML includes a TouchSensor node, Blendo applications are primarily driven by a *ButtonSensor* node. The buttons on the input devices such as PC keyboards, remote controls, game controller pad, etc. trigger this node.

```
ButtonSensor {
   field SFString buttonOfInterest "ENTER"
   field SFTime pressTime 0
   field SFTime releaseTime 0
   field SFBool enabled TRUE
}
```

In addition to the standard computer keyboard keys, Blendo has predefined a set of literal strings that are recognizable as values for the *buttonOfInterest* field. Depending on the type of the input device, these literal strings are then mapped to the corresponding buttons of the input device. For example, if the *buttonOfInterest* field contains the value of *"REWIND"*, the corresponding mapping key for a keyboard input device would translate to '¬ ', whereas on a TV remote it would map to the '<<' button.

We have taken care in designing the graphical user interface for our prototype application based on the assumption that TV viewers are typically limited to four *arrow* buttons, a *select* button, and an *exit* button. Furthermore, for the most part we are sticking to the traditional 2D style of menu-driven interface. As seen in Figure 4, the menu selections are located on the left side of the screen.

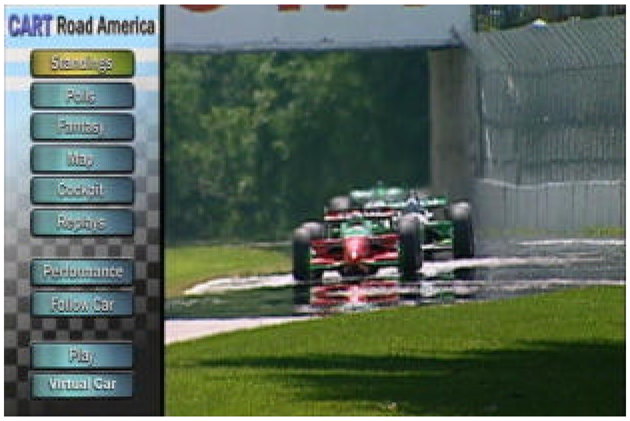


Figure 4: Screen layout for user interface.

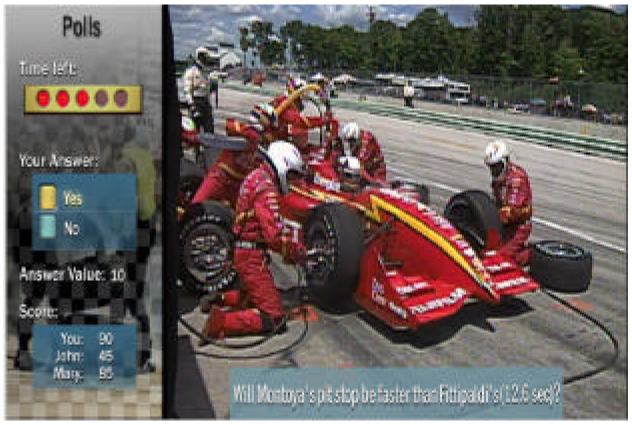
For each of the menu buttons, we have three images to represent the states of the button to provide the visual cue as to whether the button is in a *normal, highlight, or selected* state. We have implemented a script node for managing the viewer's interaction with the menu interface. It is responsible for updating the menu display and to trigger the appropriate actions when viewer has made a selection.

5. RESULTS

We have developed an initial prototype for interactive sports enhancements based on a CART race from the 1999 season. This prototype includes a set of interactive scenarios, ranging from simple statistics on demand to integration with a video game. The extensions that we focus on in this paper serve as the foundation for much of our work to date.

For our platform we used Blendo, which enables content that is portable across a variety of devices that are capable of supporting Interactive TV, including game consoles, set top boxes, and future digital televisions. Our first prototype was built on SGI 320 and 540 NT Workstations since the data paths, particularly for video texturing, were quite stable there ahead of other graphics hardware. We now have the prototype running on the PlayStation 2 game console as well, which has an excellent architecture for supporting our graphics requirements. Blendo interprets the new nodes on these platforms using the declarative representation of our extensions.

The three scenarios described in Section 3.1 are implemented in our prototype. Since DTV supports a 16x9 screen aspect ratio, we use the VideoSurface node to display video with a 4x3 aspect ratio and present additional information and menus in a vertical region on the side of the screen.



Will Montoya's pit stop be faster than Fittipaldi's (12.6 sec)?

Figure 5: Polling question with scripted timer based on triggers.

In Figure 5, we illustrate how a polling question appears using a translucent Blendo layer and a scripted timer to manage user input via the remote control. People watching the program on a non-Blendo platform (e.g., traditional TV) do not see any of these local enhancements. In this case, the data is bursty in nature, only involving synchronized triggers along with the polling question and, on completion, the results. This is similar to the triggers in ATVEF [11] that are used in current interactive TV applications. Blendo provides its own event model for processing the triggers. The ButtonSensor node is used for all viewer interactions.

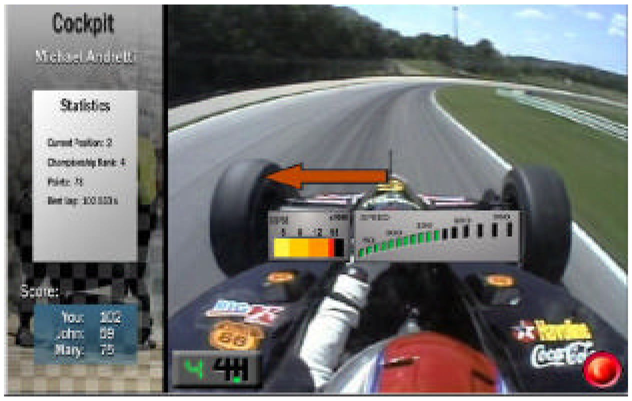


Figure 6: Animated overlay based on streamed telemetry data.

Figure 6 shows an overlay that reflects the real-time data from the car telemetry. The arrow is based on lateral force on the car and changes color from green to red based on amplitude. By moving this downstream to user control, the viewer can view it selectively. The viewer could choose certain cars to track, different consoles for visualizing the data, and when to display this data.

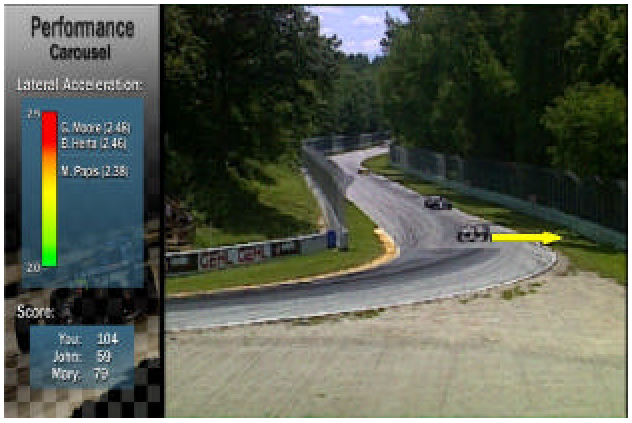


Figure 7: Registered graphics insertion using tracking data.

Figure 7 uses the same arrow glyph from the car telemetry, but it is registered with the car as it goes around a particular curve. This would be for the serious fan that wants to compare how various cars deal with subtleties of the track. This illustrates a registered graphics insertion that uses car telemetry, car position, and camera tracking data. Features like highlighting a certain car would not require the telemetry data, but would require the camera and car position information. The CameraViewpoint node maps the camera data into a virtual camera in Blendo, and the Gridnode is used to compensate for radial distortion in the lens.

6. FUTURE WORK

Our research is continuing in various areas where we can simplify the task of producing these interactive enhancements. Some areas of our current research are focusing on more flexible integration of video and graphics assets, based on a *Chromakey* node that we are developing. We also believe that video that has an associated depth map, like that used in current virtual studios, offers interesting possibilities for improving embedded graphics, so we are exploring extensions in that space. Whereas our early emphasis has been on presentation and interaction, a flexible and robust mechanism for data management is becoming increasingly important in our work, and we are currently enhancing our data management system.

Now that we have the foundation for interactive TV enhancements, we are developing some of the more feasible features of our prototype as part of an over the air test to a private set of Blendo-enabled platforms. Several stations around the U.S. are currently performing tests of their DTV systems to assess the technology and experiment with new forms of entertainment that DTV enables. We are continuing to focus on auto racing, but we are beginning to apply the technology to other sports to identify new opportunities and challenges.

7. REFERENCES

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

[2] Cavallaro, R., The FoxTrack Hockey Puck Tracking System. *IEEE Computer Graphics and Applications*, 17(2):6-12, March-April 1997. ISSN 0272-1716.

[3] Orad, http://www.orad.tv/sport/index.htm.

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

[5] The Virtual Reality Modeling Language, International Standard ISO/IEC 14772-1, http://www.web3d.org/Specifications/VRML97/, 1997.

[6] Tsai, R., A Versatile Camera Calibration Technique For High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras And Lenses, *IEEE Journal of Robotics and Automation*, RA-3(4): 323-344, August 1987.

[7] Casey, J.B., Aupperle, K., Digital TV And The PC. http://www.hauppauge.com/html/dtv.pdf, November 1998.

[8] Gibbs, S. et.al., Virtual Studios: An Overview. *IEEE MultiMedia*, 5(1):18-35, January-March 1998. ISSN 1070-986X.

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

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

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