DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences

Blair MacIntyre[1], Maribeth Gandy[2], Steven Dow[1] and Jay David Bolter[3]
[1]College of Computing, [2]Interactive Media Technology Center
[3]School of Literature, Communication and Culture
GVU Center, Georgia Institute of Technology, Atlanta, GA 30332, USA
{blair, steven}@cc.gatech.edu, maribeth.gandy@imtc.gatech.edu, jay.bolter@lcc.gatech.edu

ABSTRACT

In this paper, we describe The Designer's Augmented Reality Toolkit (DART). DART is built on top of Macromedia Director, a widely used multimedia development environment. We summarize the most significant problems faced by designers working with AR in the real world, and discuss how DART addresses them. Most of DART is implemented in an interpreted scripting language, and can be modified by designers to suit their needs. Our work focuses on supporting early design activities, especially a rapid transition from storyboards to working experience, so that the experiential part of a design can be tested early and often. DART allows designers to specify complex relationships between the physical and virtual worlds, and supports 3D animatic actors (informal, sketch-based content) in addition to more polished content. Designers can capture and replay synchronized video and sensor data, allowing them to work off-site and to test specific parts of their experience more effectively.

Categories and Subject Descriptors: D.2.2 [**Design Tools and Techniques**]: Evolutionary prototyping, User interfaces; H.5.2 [**User Interfaces**]: Graphical User Interfaces (GUI), Prototyping; I.3.7 [**Three-Dimensional Graphics and Realism**]: Virtual reality; J.5 [**Arts and Humanities**]: Fine arts

Keywords: Augmented Reality, Mixed Reality, Design Environments, Capture/Replay, Animatics, Storyboards.

INTRODUCTION

Over the past few decades, augmented reality (AR) researchers (including ourselves) have explored a wide variety of task-focused domains, ranging from equipment maintenance and repair to medicine to battlefield awareness. Over the past four years, we have been collaborating with new-media designers[1], shifting our thinking from "AR as technology" to "AR as medium," and turning our attention toward more experiential AR domains such as in-situ educational historic dramas, reflections on biased point of view in "3 Angry Men" [9], and entertainment in "Alice's Adventures" [12].

In this paper, we use the term AR to refer to a broad class of user interface techniques that seek to augment a person's perception of the world around them with computer generated information. (In our work, we tend to use see-through head-worn displays and headphones to precisely align graphics and sound with the user's view of the world around them, but the issues and solutions discussed in this paper are applicable to a broad class of systems that combine sensing with *in-situ* presentation of computer generated content.) There are significant technical challenges to creating working AR systems, but these challenges can be overcome in specific situations through careful engineering and design. When working on a well-defined AR task, such as a printer maintenance system [4], it is feasible for a team of technology experts to work closely with design and HCI experts to understand the particular problem and engineer a solution.

Unfortunately, tight collaboration between designers and technologists is not effective for less well-specified design projects, where AR is the medium being used by a designer to explore a problem. Designers are most effective when working directly with a medium, and working through an intermediary seriously hinders (or even destroys) the creative process. Consider the difference between a painter directing an assistant where to apply oil to a canvas, rather than holding the brush. Or a 3D animator describing to a programmer how a character should move, rather than manipulating the model directly.

Skilled designers are also not novices in need of hand-holding. Design tools (such as the 3D animation program mentioned above) do not have to be easy to use to be useful; rather, they need to be predictable, understandable, well documented and powerful, and to support a work style that is appropriate for the medium. Our target users are these "typical" skilled designers.

DART (The *Designer's AR Toolkit*) is the result of our collaborations with designers over the past four years, and is aimed at enabling them to work *directly* and *effectively* with AR. The design of DART addresses a collection of problems that, together, make AR a difficult medium to work with. DART is implemented on top of Macromedia Director, the defacto standard for multimedia content creation. Director was chosen because it is a full-featured, widely used tool that is relatively open and extensible, has a robust debugging and design environment, and is powerful enough for final content delivery.

[1] For simplicity, we will use the term designer to refer to those people whose primary interest is in creating experiences or artifacts of some kind; this could include graphics designers, artists, architects, game designers, media theorists, museum exhibit designers and so on.

UIST '04, October 24–27, 2004, Santa Fe, New Mexico, USA.

There are three main contributions of this paper. First, we identify a collection of problems that make AR a particularly difficult medium to work with, and suggest an approach to AR design (manifested in DART) that addresses many of these problems. This design process emphasizes rapid exploration and testing of AR experiences using informal content. Second, we argue for the importance of integrating research tools with existing commercial authoring software (Macromedia Director, in our case), and report on our experiences (especially the impact of the drag-and-drop timeline-based authoring model used by many such tools). Finally, we present a collection of novel features of DART that support our approach to AR design. These features include integrated support for physical and virtual content in the design environment, support for sketch-based animatic content, and capture/playback facilities that break the need for working in real-time in the target site.

AR IN THE WILD: DEATH BY MANY SMALL CUTS

In this section, we highlight the problems that have made AR a difficult medium to work with, and briefly summarize how DART addresses them. Many of these problems are familiar to AR researchers, and some have been discussed or investigated elsewhere. However, no other AR environment has attempted to provide a comprehensive solution to these problems.

We group these challenges into three broad categories: *design and programming*, *enabling technology*, and the implications of *working in the physical world*.

Design and Programming

There are no simple, flexible programming environments for AR. Designers are accustomed to tools such as Macromedia Director and Visual Basic, which mix visual layout, direct manipulation and scripting. In these environments, simple experiences can be created with little or no programming, and complex systems can be created by delving into the scripting language. Visual layout is used when appropriate (e.g., positioning 2D elements, organizing content on a timeline, etc.), but not for everything. Since there are many things that cannot be specified graphically (e.g., complex control flow and interactions among multiple elements over time), most designers learn to program (textually) out of necessity when working with these tools. This is not a bad thing (there are many things that are most concisely expressed with program code), and allows us to assume that our designers can and will venture "into the code."

DART is integrated into Macromedia Director, and designed to complement the common development style used by Director developers. We assume designers will continue to use their existing tools (e.g., Photoshop, Maya) for content creation.

Programming must be done at too low a level. Most AR tools are designed by computer scientists for computer scientists (e.g., Studierstube [15], the ARToolkit [2]). They require the use of programming languages such as C or C++, and combine general-purpose libraries such as OpenInventor with (occasionally poorly documented) custom libraries. These languages and libraries are a poor match for the work practices and talents of typical designers, who are skilled with visually-oriented design tools (e.g., Flash, Photoshop or Maya). While these tools often have integrated scripting, most designers do not have substantial (if any) programming experience in languages like C++ or Java.

The bulk of DART's functionality comes from a collection of *behaviors* (structured scripts) written in Lingo, Director's interpreted scripting language. These behaviors can be easily integrated with other Director content, and can be modified at will, allowing DART to be extended by designers as needed.

3D content is expensive and time-consuming to create. AR content must be integrated with the physical world, and is thus inherently 3D in nature (even if some of the individual elements might be 2D graphics or audio). As Landay and his colleagues have shown (in a variety of domains), the key to supporting creativity and design exploration is to encourage a gradual transition from informal (i.e., sketched) to formal content as the design is explored and refined (e.g., [8]). We view 3D content creation as one step in the shift from initial design ideas and storyboards to working experiences, and (following Landay) believe it is important to support sketched content during early design. Skilled designers can create aesthetically pleasing and emotionally evocative content using 2D sketches, (e.g., Figure 8(b)), far beyond what they could do in a similar amount of time with 3D modeling tools (including research tools, such as Teddy [7], that have been designed to "simplify" the creation of 3D content).

DART supports rapid creation of informal content from 2D storyboards via *3D animatic actors*. Animatics are sequences of 2D storyboards synchronized with audio, often used during film previsualization. *Animatic actors* are sequences of sketches used as placeholders for individual content elements, supporting rapid assembly of, and experimentation with, an experience prototype prior to final content creation.

Enabling Technology

Dealing with multiple, unrelated tracking technologies is extremely difficult. Spatial trackers are one of the fundamental technologies needed for AR experiences. Most available technologies, from computer vision-based marker tracking to devices with physical sensors, are difficult to set up and use. Many libraries have been created to simplify the use of these technologies (e.g., the ARToolkit [2] and VRPN[2]). However, these libraries are themselves hard to use, and are difficult to integrate and use together. For example, creating an experience that integrates vision-based sensing with other tracking technologies, or substituting one kind of sensing for another as development progresses, is currently a daunting task, even for skilled technologists.

DART provides uniform access to these sensors, letting them be mixed and matched as desired. The code that integrates the different sensing technology is centralized in the Lingo code, allowing it to be modified as needed for specific situations designers may encounter.

Sensing and reasoning technologies are expensive (in time and money) to create and deploy. Most interaction in AR is implicit and dependent on the specific applications (e.g., is the user looking at the statue? Did the user open the lid on the machine?). Thus, in addition to commercial tracking systems, AR systems often require custom sensing and reasoning to be implemented. As with 3D content creation, experience testing and design is seriously hindered by the difficulty and time required to design, create and deploy these technologies.

[2] http://www.cs.unc.edu/Research/vrpn/

DART is designed from the lowest level to support Wizard-of-Oz simulation of sensors and experience logic. (We do not address Wizard-of-Oz simulation in this paper, as we have not yet explored this DART functionality in depth).

No separation of concerns between system components. Since most AR toolkits force the programmer to work at a low level, the implementation of an AR experience typically ends up mixing high- and low-level concerns (e.g., experience scripting and tracking issues), and thus being intimately tied to the technologies used during design. This lack of abstraction makes it difficult to switch to new technologies as they become available.

DART is designed to encourage modular experience development by having each actor (content element) defined as an independent entity that is indirectly linked to other elements and to sensing and tracking hardware.

Working in the Real World

Managing relationships between the physical and virtual worlds is difficult. In AR, objects in the physical world make up most of the content of the eventual experience. However, current AR systems provide little or no explicit support for managing knowledge about the physical world, and the relationships between virtual and real objects.

DART supports a mixture of physical and virtual content. Physical content can range from elaborate models of physical structures to 3D points in the world. Virtual objects can be positioned relative to physical ones, physical objects can obscure virtual ones, and both can interact using the underlying physics engine.

Having to actually work (develop) in the physical world can be prohibitively difficult. An often-overlooked impediment to developing AR experiences is the need to be physically present during the development cycle in the environment being augmented, and to get up and move around the physical space during testing. This is especially painful for outdoor experiences, where factors such as weather, a lack of ergonomic work areas, and the lack of power and networking hamper effective development.

DART supports the synchronized capture and playback of video and sensor information, allowing designers to work anywhere they find convenient, testing their experience against representative snippets of captured data.

Working in real-time is difficult. When many things are happening in a split second, and when interactions are based on possibly noisy sensor data, debugging and understanding an experience can be difficult or impossible.

All time-based scripts in DART are based on the *DARTClock,* an abstract wrapper around the Director clock. This wrapper allows the designer to control time by pausing the experience time, changing its speed, or moving the time backwards or forwards to a specified point. While live video and sensor data cannot be paused or stepped through, the rest of the experience can. When recorded data is being used, everything (video, tracking and actor scripts) can be paused and stepped through at whatever pace the designer needs.

TOWARDS A DESIGN PROCESS FOR AR EXPERIENCES

The goal of our research is to create tools that allow a wide range of designers to explore the medium of AR. Integrating AR technology (e.g., video cameras, computer vision and a wide variety of physical sensors) into a sufficiently powerful authoring environment such as Director is a first step toward this goal, but is not sufficient. As illustrated in the previous section, there are many other reasons that creating 3D experiences which blend physical and virtual worlds is difficult.

Schell and Shochet have shown (in the context of mixed-reality theme park rides) that design ideas for experiences that integrate physical and virtual worlds must be tested in the target site as soon and as often as possible [14]. By addressing many of the problems faced when working with AR, our design for DART takes a significant step toward supporting rapid *in-situ* design and testing of AR experiences, which we believe will be the foundation of an effective AR design process. In particular, DART enables (and even encourages) in-situ experience testing by combining informal, animatic content with synchronized capture/playback of video and sensor information.

Animatic content allows design storyboards to be moved into the 3D world quickly, and to be quickly changed as the design progresses. Synchronized capture/playback supports experimentation, refinement and testing of design ideas in the context of the target environment, without needing to work *in* the target site. Capture/playback actually encourages in-situ testing because it is typically easier to work with captured data (that delivers synchronized, realistic sensor and video information to the evolving prototype) than it is to work without such input. We expand on these ideas later in the paper.

MOTIVATION: A LOT OF PAIN FOR A LITTLE GAIN

The history of media has shown that any medium (not just AR) will not reach its potential until it is put into the hands of designers who, through their work, eventually define the popular forms of the medium. Gutenberg invented the printing press, but not "the novel". Edison invented motion pictures, but not "the film". Berners-Lee invented HTTP and HTML, but not "the web" as we now know it. And so it will be for AR. Sutherland invented AR technology, but we do not yet know what forms the medium will take on over time.

Our desire to push forward the development of AR as a medium, by putting it in the hands of designers, did not arise in a vacuum, but from our experiences over the past dozen years of AR research, and more recently from four years teaching a multidisciplinary course on *AR Experience Design*. While the lack of decent tools during our early research in AR was annoying, it was a severe problem for our design colleagues and students. We noticed that, while these designers usually produced polished demonstrations of their ideas in other media, their work with AR was often far less technically and conceptually refined, even when we integrated the necessary AR technology (e.g., tracking and live video) into Director and had experienced AR technologists work with them. Closer examination revealed that the difficulty of working with the technology, combined with the absence of support for traditional design activities (such as rapid prototyping and design exploration), was seriously impeding their work with AR.

Our experience is not unique, and perhaps (when you consider the state of the technology) not particularly surprising. For example, while there have been a growing number of conferences devoted to creative uses of new technologies such as

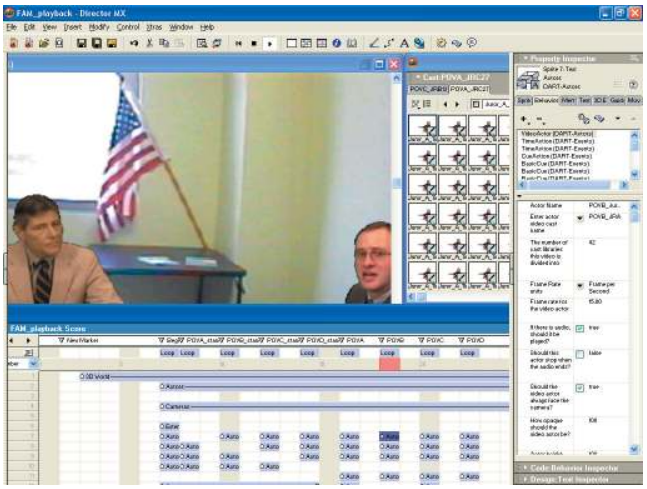

Figure 1: An example work session in DART (while debugging the Four Angry Men (FAM) experience). The entire score for FAM is visible, including the nine scenes and most of the actors (each scene is a column in the score). The stage (containing the running experience) is visible, as is part of the content for one video actor, and some of Director's editing windows.

AR and mixed reality (MR) (such as CAST, TIDSE, IWEC and ACE) many of the experiential projects reported at these conferences never make it past the theoretical or early design stages into working prototypes. In both our work, and in the wider research community, instances of successful design projects using media like AR have usually been slow, painstaking endeavors carried out by close-knit teams or technically sophisticated individual designers (e.g., [5]).

A Motivating Example: *Alice, Three and Four Angry Men*

While these occasional successes are encouraging, they rarely provide a firm foundation for future work because the implementations usually mix low- and high-level details and become intimately tied to the problem solved. For example, the designer who implemented our *Alice* system [12] in Director helped us use his code to do a proof of concept for *Three Angry Men* (TAM), a dramatic AR experiment in point-of-view [9]. TAM is a re-creation of the movie "12 Angry Men," where the participant can move around a table, occupy the positions (and minds) of the jurors discussing a court case, and see the drama unfold as colored by each juror's biases. Even with his help, completing the initial implementation of TAM (technically, a very similar experience to Alice: a group of video-based actors seated around a table with the participant) took many months, much of it devoted to content creation. Unfortunately, when the first version of TAM finally worked, we discovered a variety of problems with the design; fixing the design required us to recreate all of the content.

The update of TAM, now called *Four Angry Men* (FAM), was created in parallel with DART over the past year, and was our first major test of DART. We completely reimplemented TAM without using any code from the original prototype (except where TAM code had made it into DART itself). The resulting prototype ended up being simple and concise, and required only a small amount of custom code to deal with the specific interaction techniques of standing and sitting to pause and resume the experience. We used animatic actors to develop and test the experience before the content was complete. The screenshot of the FAM prototype in Figure 1 shows the entire score for the experience along the bottom. While FAM does not represent a formal evaluation of DART, nor does it exercise all of DART's functionality, it does represent one example showing that complex AR experiences can be implemented with our system. The final content for FAM is almost complete, and represents over a year of work in itself.

HOW WE LEARNED TO STOPPED WORRYING AND LOVE DIRECTOR

When we began conceiving of this project, we made a decision to integrate whatever we did into existing tools, rather than create our own, so that designers would not have to learn a new tool or use crippled "research grade" tools. This last point should not be minimized: research systems are often weak in the areas that are farthest "from the research," and it is in those areas that commercial systems typically excel.

We chose Macromedia Director as the basis for DART. Director is a flexible, general-purpose authoring tool for Shockwave content (the 3D component of which is Shockwave3D) and is widely used for CDRom, desktop and kiosk multimedia development. Director has a rich feature set, an interpreted programming environment called Lingo, and a freely available plug-in SDK (allowing us to extend it as necessary). Other popular platforms that we considered (but eventually rejected) include Flash (it does not have native 3D support, has weak support for non-vector-based content, has a weaker scripting language than Director, and does not have a freely available plugin API) and MAX/MSP (it uses a graphical data-flow model that makes it difficult to develop narrative content and requires designers to develop new functionality in C/C++).

Most interaction designers, including the students and researchers with whom we work, are skilled with Director. There is a vast amount of existing support for Director that we can leverage: hundreds of books, Xtras (plug-ins) for everything from low-level OS access to database integration, official and unofficial web sites, mailing lists and discussion groups with an overwhelming amount of sample code, and so on.

Furthermore, while it is possible to create visually interesting content without programming in Director, virtually any interactive Shockwave content requires at least simple Lingo programming. Therefore, experienced Director users are often surprisingly good programmers. Macromedia designed the bundled Lingo scripts with this in mind; they are all well documented and designed to be instructive of how to do "typical" things with Lingo. It is common for designers to appropriate scripts and modify them to suit the task at hand.

We adopt the same approach with DART: our entire collection of Lingo scripts is designed to serve as an example of how to create an AR system, as much as it is a complete AR system itself. It is our intention (and expectation) that designers will pick and choose what parts of DART they use, and modify those parts to suit their needs. Our behaviors are designed to be used alongside other Lingo scripts that a designer may have already developed, and to be modified, copied, rewritten, and extended by designers as needed.

As with other non-AR researchers who have extended Director to support their collaborations with artists and designers, we do not assert that it is the ideal environment for our work. For example, Shockwave3D does not support stereo displays, and requires us to use a very simple 3D camera model. We

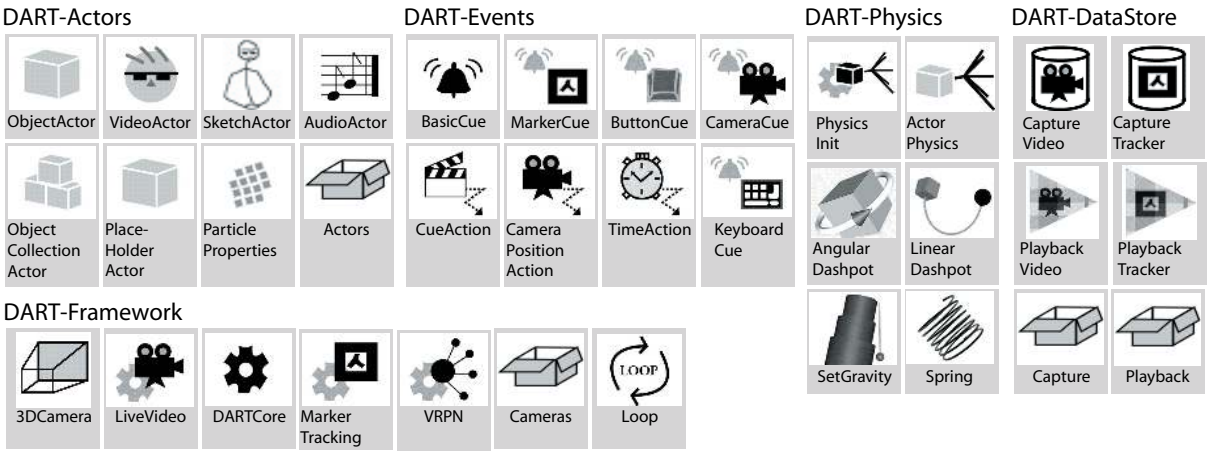

Figure 2: The different categories of DART behaviors. Each of the groupings is stored in a separate Director cast, and represents a separate part of DART. The DART-Framework provides the lowest level layer of abstraction, wrapping up the functionality in the DART Xtra and providing other basic runtime services. The empty box icons are the text sprites used as proxy objects (containers) for the DART behaviors.

can circumvent the former problem using two machines and an external video multiplexer, but the simple camera model limits the accuracy of 3D registration in some cases. The environment is strictly single-threaded, which can impact performance. Finally, the timeline-based structuring metaphor is not a natural metaphor for interactive content, as we discuss in the next section. However, it is a very good system in many ways, especially its support for a mix of visual and textual programming, and it is sufficiently powerful to deploy working AR systems (as our prototypes have demonstrated).

DART IMPLEMENTATION

The challenge of designing DART was to create a system that matches the authoring style commonly used for complex Shockwave content by experienced Director developers, yet is appropriate for AR and meets our research goals. We informally interviewed experienced developers, looked at examples of complex content, and spent many months monitoring the Director-games and Director-3D mailing lists to become as familiar as we could with common Director work practice. In the end, the DART model is similar enough to the common Director authoring styles that, if you are experienced with Director, you are already familiar with the programming model used by DART.

In this section, we will give a (necessarily terse) overview of Director and how DART is integrated into it. In the next section, we will focus on those features that are either novel, or provide a solution to one of the problems mentioned earlier.

Director Development Overview

Director includes an object-oriented language called Lingo and is based on a stage production metaphor. The environment includes a *stage* (where content is placed), multiple *casts* (where all content elements are stored, including images, video, 3D content, Lingo scripts, text data and so forth), a *score* (the timeline of the experience) and *sprites* (cast members that have been placed on the stage or in the score). Lingo scripts (typically called behaviors) are interpreted and can be attached to cast members, the stage, sprites or frames in the score; Director automatically generates graphical interfaces for editing behavior properties based on structured comments in the script.

The main structuring mechanism in Director is the score. When an application runs, the "play-head" moves across the score from left to right. Since interactive applications do not typically follow a fixed linear script, Director developers mainly use the score as a method of visually creating single-frame logical sections through which the program execution will flow, and loop on those frames (see Figures 1 and 4).

A designer places sprites on the score and can add behaviors to them by dragging behavior scripts on to the sprites. Scripts can also be placed on individual frames, or on an entire application. In a 2D application, a sprite will be a graphical component that will appear on the stage. Behaviors placed on a sprite will typically be used to modify the sprite's appearance or its reaction to inputs such as mouse buttons.

The Director environment follows a playback paradigm where an application can be played, stopped, and rewound. When the play-head enters a frame containing a sprite that component becomes active, causing the *beginSprite* function in any behaviors attached to the sprite to be called. When the play-head exits the frame the *endSprite* function is called in all behaviors. Similarly, *enterFrame* and *exitFrame* are called on each behavior when a frame of the score is entered and exited, respectively. A common programming technique is to place a script on a frame that causes the play-head to loop on that frame, and then use the *enterFrame* and/or *exitFrame* functions as the "main loop" of the program. Events are then used to cause the play-head to jump to the different frames in the score that represent logical sections of the application.

The stage represents the output of the application, and developers can directly manipulate their application components in this area. Typically, developers use a combination of graphical programming and scripts to create and manipulate elements on the stage; anything that can be done graphically can be done from scripts (but not vice-versa). Since all of the behaviors are written in Lingo and editable by the user, it is common for developers to modify the standard behaviors while also writing their own from scratch. This approach allows for rapid application development via direct manipulation of sprites, while also allowing for advanced programming via scripting. The use of the score produces a visual representation of the application that can be easily changed while testing by simply

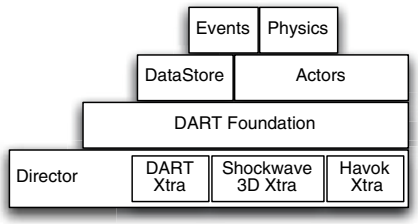

Figure 3: The relationship between the major pieces of DART. Each piece can be used independently from the pieces in the same or higher layers.

moving elements on the score and dragging behavior scripts on and off of sprites. Sprites and scripts can even be modified during run-time, which aids in debugging and prototyping.

As with any Director application, an experience created using DART will consist of a series of sprites on the score populated with behavior scripts that define the properties and interactions of the sprites. The initial set of DART behaviors is shown in Figure 2, and their relationship to each other, Director and the DART Xtra is shown in Figure 3. An example of the score for a DART application is show in Figure 4 (a screenshot of that application is shown in Figure 5).

Container Sprites: Integrating 3D content with the Score

In DART, the stage is covered by a single Shockwave3D sprite representing the 3D world. A significant problem that we needed to engineer around is that 3D content is not well integrated into the Director model (Shockwave3D is actually an Xtra). A 3D world is represented by a single Shockwave3D cast member, and corresponds to a single sprite on the score; the content elements of the 3D world are not individually represented on the score. The content of Shockwave3D sprites can only be accessed via Lingo scripts.

We get around this problem by creating sprites of other media types (currently, we use text items), and hide these text elements under the Shockwave3D world (this happens automatically, as Shockwave3D sprites usually sit above other Director sprites for performance reasons). By attaching DART behaviors to these *container* sprites, they take on the desired AR-specific functionality. We leverage the *beginSprite*/*endSprite*/*enterFrame*/*exitFrame* methods (see the previous section) to make these container sprites behave appropriately. For example, sprites representing 3D content initialize themselves and add their content to the 3D world in the *beginSprite* method, and update the state of the 3D objects in the *enterFrame* method. DART has behaviors to copy 3D content from other Shockwave3D cast members, create new content using Lingo commands, and implement DART-specific objects such as video-based actors and animatic actors.

By using container sprites, DART allows the developer to leverage some of the benefits of Director's score in the 3D domain. The designer can create sprites and drag them in and out of specific frames (and thus in and out of the experience). Extra content can be created and stored in frames that are never entered during the experience flow. Quick "what-if" tests can be performed by copy-and-pasting a collection of sprites (and thus all of their behaviors) to a new location, making a few changes to the behavior properties, and then trying out the experience in that frame. The DART behaviors also make heavy use of Director's automatically-generated property pages to

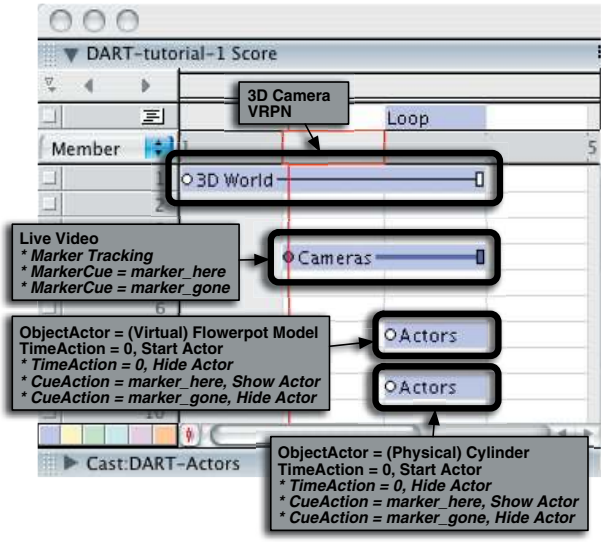

Figure 4: The score for a basic DART example. A screenshot of the running experience is shown in Figure 5. *Loop* is a script that tells Director to repeat this frame forever. *3D World* is a Shockwave3D sprite that covers the stage. *Cameras* and *Actors* are text sprites that are hidden under the *3D World*, and serve as containers for DART behaviors. The behaviors attached to each sprite are listed in the inset boxes, with some key parameters. The italicized behaviors with an asterisk at the start of the line would be included when using marker tracking; if global tracking (via a physical head- or camera-tracker) is used, they are not needed.

allow the designer to set behavior parameters, such as the local transformation of a graphical object, an object's parent in the scene graph, and the linkage of an object to incoming tracker data. Behaviors can be added to a sprite to tell the object when to create or respond to events. Sprites are also used to hold behaviors for configuring trackers and cameras, and to control video and data capture and playback.

Container sprites can be assigned any names the designer chooses, although we provide a few predefined containers in the DART casts for simplicity. In Figure 4, the *Cameras* and *Actors* sprites are "container" sprites.

The DART Xtra

The foundation of DART is the *DART Xtra,* a Director plugin written in C and C++, that provides an interface to the technologies needed to work with AR. These include real-time live video capture (including support for DirectShow and custom capture libraries from Point Grey and Videre Design), the ability to feed a live video stream into the texture memory of a Shockwave3D scene (to support video-mixed AR, where the video stream is displayed in the background of a 3D scene), support for visible marker tracking on live video streams (currently using the AR Toolkit) and an interface to a wide collection of other tracking and sensing systems (to track the user's head and other objects, using VRPN). From VRPN, the DART Xtra also gets a basic distributed shared memory system that allows integers, floating point numbers and strings to be replicated reliably across any number of processes.

The DART Xtra is not typically accessed directly by the developer. Instead, the DART-Framework behaviors provide structured access to the DART Xtra functionality. The DART Framework consists of behaviors that give the designer

Figure 5: A screenshot of a running DART program. The "program" for this example is shown in Figure 4. There are two objects, the virtual flowerpot, and a "physical" cylinder that is aligned with the cup (hard-coded relative to the fiducial in this example) and used to cause the cup to interact correctly with the flowerpot.

control over the 3D camera in the Shockwave3D world (*3DCamera)*, the camera parameters for live video capture (*LiveVideo)*, whether to do marker tracking on the video stream (*MarkerTracking)*, and the VRPN tracking and sensing system (*VRPN)*. The *3DCamera* behavior allows the designer to use its property pages to set its linkage to the scenegraph, such as connecting it to a tracking device or not . Examples of property pages for these sprites are shown in Figure 6.

The *DARTCore* movie script (a "movie script" is a global script) handles application-wide services such as the subscription to and distribution of incoming tracker data and events, and the global abstract clock (*DARTClock*) mentioned earlier. All DART behaviors use the *DARTClock* rather than the Director clock, so the designer can exercise complete control over the "time" used by the DART behaviors.

KEY FEATURES OF DART

At one level, DART represents a design and engineering effort that integrates a carefully chosen set of AR authoring concepts into the Director authoring model. DART uses the score for content organization in a way that is common among Director developers, and most of the system is implemented in the interpreted language, allowing it to be edited as necessary. DART combines the Director sprite and event model with an event-based actor model common in VR systems. DART has a set of events that are similar to any complex VR or AR system (e.g., [3] or [15]), as well as additional events geared towards the set of actors currently supported. Multiple levels of abstraction are provided to the low-level technology (all of which can be used by the designer): a direct interface to the Xtra, a more structured global interface in *DARTCore*, and a high-level interface through the actors.

In addition to careful design and engineering, however, there are a number of interesting and novel elements to the design of DART which we discuss in the remainder of this section.

Actors for AR: Physical/Virtual Interplay and Animatics

DART currently supports 3D models, audio, video-based content, and animatics. The DART-Actors behaviors are used to define these content objects that make up an AR application. Common properties, such as local transformations, texture

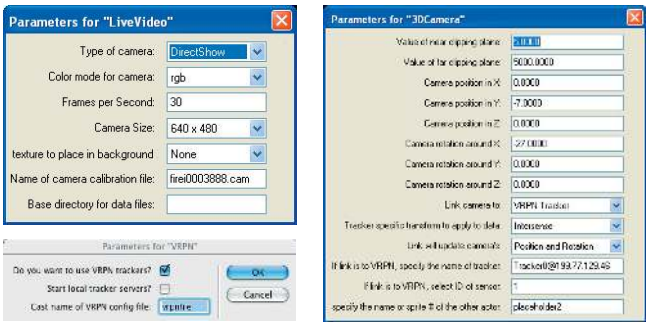

Figure 6: Examples of property pages created by Director for the DART behaviors. *LiveVideo* allows the parameters of a video camera to be specified (the driver, size, speed, camera calibration data filename). *VRPN* allows the VRPN subsystem for tracker access, as well as servers within the DART Xtra, to be enabled or disabled. *3DCamera* controls the parameters of the 3D virtual camera, including attaching it to trackers or actors.

and color, placement in the scene graph, and linkage to trackers can be specified via property pages. The specific set of actors included with DART were created based on our needs; to create a new actor, one of the existing actor scripts can be copied and modified as needed (e.g. one of the students in our AR Design class developed a spatialized audio actor based on the *ObjectActor*, the *AudioActor* and a spatialized audio Xtra). The new actor must support a basic set of functions, such as "*StartActor*" and "*MoveActor*," and any of the actions the developer needs.

The *ObjectActor* is used to add a 3D object to the world, either a primitive object (such as a cube or cylinder), or an imported Shockwave3D model. A novel feature of the *ObjectActor* is that the objects can represent either "virtual" or "physical" content. A virtual object is rendered normally and appears in the scene. A physical object, on the other hand, is used to model known objects in the physical world. The physical objects are rendered into the z-buffer (but not the color buffer) before rendering the virtual objects, allowing physical objects in the world to appear to occlude virtual objects, as shown in Figure 5. Real and virtual objects can interact using a physics simulation that includes collision detection, inertia, gravity, and constraints. The DART-Physics behaviors are used to specify the physics properties of actors. The physics simulation is handled by an Xtra containing a basic version of the Havok[3] Physics Engine (included with Director).

The *VideoActor* texture-maps video onto a polygon in the 3D world, and plays a synchronized audio track if one is present. We leverage Director's highly efficient data management facilities and store the video as individual 32-bit RGBA image cast members in a Director cast, with the audio track stored in an audio cast member in the same cast. While this takes a significant amount of space (e.g., the video actor content for FAM is over 5 gigabytes in size), it allows the frames to be played back rapidly with alpha information (for transparent backgrounds) intact. Frames of a video actor can be seen in the cast in Figure 1.

The *SketchActor* implements animatic characters, a key element of our support for rapid design exploration. Technically, *SketchActors* are similar to *VideoActors*, but rather than

[3] http://www.havok.com/

displaying fixed frame-rate video, a *SketchActor* flips through a sequence of sketched images, with the exact timing information specified by the designer. A cast for a *SketchActor* contains the sketches (a set of image cast members), a text cast member with the timing information, and an optional audio track. A snapshot of a scene containing three early animatic mock-ups of the jurors in *Four Angry Men* is shown in Figure 7(a). The final content, using *VideoActors*, is shown in Figure 7(b).

Synchronized Capture and Playback

The second key feature of DART is support for synchronized capture and playback of video, tracking and other sensor data. We have found, in our own work and in our discussions with others, that the inconvenience of working with real devices (trackers, head-worn displays, cameras, etc.) in a real physical location (an historic site, a lab with equipment being augmented, outside in a city, etc.) is a significant impediment to working with AR. Having to be in the physical space, with the physical devices set up, to test design ideas and debug prototypes, is often impractical. Therefore, AR designers often develop "blind" and avoid testing until the last minute. Wizard-of-Oz simulation of sensors can help with this problem [6], but manually simulating one or more 6 degree-of-freedom spatial trackers is difficult, and does not address the need to see the graphics merged with the physical world.

The DART-DataStore behaviors allow designers to capture and playback video and tracker data. The captured data is stored in a Director cast (as a collection of images for the video frames, and separate text cast members for each stream of sensor data). Playback is transparent to the other DART components; the data is inserted into the system so that most of DART cannot distinguish between live and captured data. Therefore, nothing has to be changed in the behavior properties to switch between live and prerecorded data. Switching from one to the other is as easy as dragging Playback sprites and live Video and Tracking sprites on and off the score.

To capture data, the developer adds *CaptureVideo* and/or one or more *CaptureTracker* behaviors to the score. Video can be captured at a lower frame rate and size than the original, to save space during long capture sequences. A separate behavior is used for each tracker to be captured, with each different fiducial marker being treated as a separate tracker, allowing the desired set of sensors and markers to be saved. Typically, we capture video, multiple trackers and data from multiple fiducials simultaneously.

PlaybackVideo and/or one or more *PlaybackTracker* behaviors are added to the score to play back captured data. The playback behaviors respond to cues to control when they start, stop, pause or restart playback. Captured video and live video cannot be used together, but captured marker tracking data and VRPN tracker data can be mixed transparently with live tracking. A key feature of the playback behaviors is that they are based on the *DARTClock* (DART's global clock, mentioned earlier). The playback of video and sensors works correctly as the clock is paused, fast forwarded, rewound, advanced at variables rates, or even set to arbitrary values. The combination of synchronized playback with a controllable clock allows designers to examine their experiences at speeds other than real-time, an invaluable design and debugging facility.

Captured data can be used for more than offsite development. For example, the movements of a tracker can be treated as simple motion-capture data, and used to control animated objects in the scene. (To support this, *PlaybackTracker* replays the data "absolutely" or "relatively", where "relative" playback reports each data value relative to the first data value).

(a) Animatic scene (b) Final Scene

Figure 7: Captured scenes from *Four Angry Men*. The animatic scene is shown on a table in our lab. The final scene is in a conference room set up to look like a jury room.

Event-based Programming Using Cue and Actions

DART provides an AR-specific event system for the DART Actor objects. The DART-Events behaviors are split into *cues* (named events that are broadcast when some condition arises) and *actions* (actions that occur in response to a named cue event or some basic condition), and support a mixture of common VR-style and DART-specific events. For example, a cue could fire based on the distance of one actor to another in 3D space, a particular frame in the *VideoActor* being played, a specified fiducial marker coming into view, or a key being pressed on the keyboard.

To create a cue, the designer places the appropriate cue behavior on an Actor, Camera, or the 3D World, specifies the name of the cue that should be fired, chooses the condition that will trigger the cue from a drop-down list (e.g., the collision of this actor with another, or a position in space), and specifies any additional required parameters (e.g., the actor to watch for collision with, or the x,y,z position). The *CueAction* behavior is used to perform an action in response to a cue. When a *CueAction* behavior is attached to an Actor, the designer specifies the cue to respond to and the action the Actor should perform (e.g., starting an audio clip, moving the actor, moving the play head to another location on the score).

In addition to the *CueAction*, we have found it onerous to create pairs of cues and actions for the most common actions (time-based actions that fire when a specific time is reached in the active frame, and actions that fire based on the location and orientation of the virtual camera). Therefore, DART has two behaviors, *TimeAction* and *CameraPositionAction*, that fire actions in response to these events without needing to create a separate cue.

Adding new cues and actions is straightforward. The action and cue list definitions are embedded in the code in two places (the string describing the list in a property sheet, and a case statement that reacts to the corresponding property value); by adding new entries in two places, and adding a method to handle the action in each actor behavior, a new action can be added to the system and reflected in the property pages.

Handling of Tracking Data

There are two interesting aspects to the way DART handles tracking data. First, the marker tracker (that tracks ARToolkit markers in the live video stream, of the sort shown in Figure 5) reports the marker positions in world coordinates, rather

than relative to the camera (as is done in most marker-based systems). Therefore, regardless of how the video camera is positioned, either absolutely, relative to another actor or relative to a tracker, the objects attached to markers are automatically positioned in a useful, stable coordinate system. This difference is easier for designers to understand (i.e., why would anyone care where a marker on the table is in relation to the camera?), and also manifests itself when a marker is lost by the tracking system, as the objects attached to it are then fixed relative to the world rather than relative to the camera (which is usually attached to the user's head). Having "untracked" objects fixed in world coordinates is a much more useful default behavior than having them stick to the screen.

Second, DART allows designers to specify application-specific transformations to apply to the reports from different tracking hardware before the reports reach the application. Different devices report in different coordinate system, some of which are very inconvenient to use. For example, some devices report in a Cartesian coordinate system with Z up, X forward and Y to the side relative to an upright person's head, while other systems report with Y up, Z forward and X to the side. GPS units report in longitude, latitude and altitude, or in meters relative to the global UTM grid. By adjusting the data before it reaches the application, a designer can specify her experience content in meters relative to a known location at her site. Or, when moving from one technology to another, the tracking data can be adjusted instead of modifying the application content and logic. These data transformations can be specified on a per-subscriber basis, so each script in DART that subscribes to a specific tracker can arrange to see the tracker reports differently.

EXAMPLES

We have been using DART for the past year in our own work. The largest project we have used this version of DART on, as discussed above, is *Four Angry Men*. We also used a preliminary version of DART to support our AR Design class during Spring 2004. The capture/playback facilities proved invaluable in this context, as all 35 students could work on AR projects (individually and in groups) without needing to timeshare our AR equipment. For some assignments, the students never visited the target environment, but created an experience and tested it against a collection of captured data sets. The students did three AR projects using DART, plus a major experience design project. The animatic content allowed them to focus on the design of their experiences, without getting bogged down in content creation. Some groups used DART as is, and others extended it to meet their needs (to support spatialized audio, or to support Phidget sensors and actuators [6]). Example content from our class is shown in Figure 8.

We are currently using DART for a variety of other projects, including AR tours of Atlanta's Historic Oakland Cemetery (the focus of the projects in our class, and of an ongoing research project), a distributed AR game loosely modeled on the popular *Warcraft* game series, and to create the "task" for an experiment evaluating the utility of AR for poultry inspection. We have recently released DART to a small number of designers around the world, and anticipate a wider release before this paper is published. Initial feedback has been very positive, although at this point we have nothing beyond anecdotal evidence of DART's utility.

(a) Working with recorded data (b) An animatic ghost

Figure 8: (a) A TA for our AR Design class, working with DART in the comfort of a lounge outside our lab. He is grading an experience using a 30 second video and tracker segment that was captured at Oakland Cemetery in downtown Atlanta. The dataset includes InertiaCube2 orientation data and differential (WAAS) GPS data. (b) An example of an animatic ghost created by a student in the class (Image courtesy of Smitha Prasadh).

RELATED WORK

Most work in the AR community has focused on improving tracking and display technology. There have been a number of AR toolkits developed over the years, the best known of which are Studierstube [15], the ARToolkit [2], Tinmith [13] and DWARF [11]. All of these tools require the application developer to work at a fairly low level, with languages like C or C++. Our own previous system (Repo-3D [10]) used a high-level interpreted language, but (like the other systems) provided no graphical design environment. None of these systems attempt to address the wide range of issues addressed by DART. Of these, the most widely used is the ARToolkit, both because the technology requirements for using it are modest (it only supports video marker tracking, and one requires inexpensive web cameras), and because one of its sample programs shows how to attach VRML models to markers, allowing designers to explore AR by editing VRML text files.

More recent projects have begun to tackle the other end of the design spectrum, exploring environments to simplify the creation of AR systems with minimal programming. APRIL[4] (a scripting environment built on Studierstube) is an XML-based language for authoring AR, but is still preliminary and does not yet provide the breadth or power of Director and Lingo. AMIRE is focused on creating graphical authoring tools for specific AR domains, rather than on creating a general purpose environment. When working in one of their domains, on a task that one of their tools supports, their tools would probably be easier to use, but designers are limited to exactly the kinds of problems they have created tools to solve [1]. CATOMIRE[5] is a graphical authoring environment that uses a dataflow approach, where their system components are hooked together with "wires." As with AMIRE, their system is currently very limited in scope, and would limit designers to working on problems they have built components for.

Other relevant work can be found outside the AR community. The Phidget toolkit [6] aimed at making tangible devices available to designers (via COM and ActiveX), and solved some fundamental problems with prototyping beyond just "making it easy to get at the hardware." Their domain was different, however, and the scope of their work more focused.

[4] http://www.studierstube.org/april/

[5] http://webster.fhs-hagenberg.ac.at/staff/jzauner/CATOMIR.html

There have been numerous research systems created over the years to support the exploration of a new electronic medium by novice programmers with an eye toward developing better interface and programming technologies. The closest to our work is Alice [3], although they are focused on lowering the threshold of entry into the world of 3D graphics and VR programming (a much simpler medium to work with), rather than explicitly supporting early design activities. Most such systems take the approach of creating a custom programming environment, rather than enhancing an existing environment.

The motivation for our sketch-based content comes from the work of Landay and his collaborators at CMU and Berkeley (e.g., [8]). Their work focuses on supporting designers in the manner we hope to, although they have tended to work in more traditional domains.

ONGOING AND FUTURE WORK

We are in the process of finishing the first public release of DART. After this release, there are a number of directions we hope to move in. Technically, we plan to enhance the camera infrastructure to support multiple simultaneous cameras, camera calibration from inside Director, a plugin architecture for image processing operations, better multi-sensor integration, and automated multi-sensor calibration.

Another major research direction is Wizard-of-Oz (WoZ) support. DART contains the necessary building blocks for distributed WoZ interfaces, but we have just begun to explore the potential for WoZ interfaces to AR experiences. We have created a simple WoZ tracker, where a map could be placed on a tablet, and a "tracker" would send events corresponding to the pen location. However, this is only the tip of the iceberg. We believe the structure of our behaviors will allow us to generate simple WoZ interfaces automatically on the fly, supporting remote monitoring and control of our experiences.

A final direction is to tackle the problems of authoring and debugging. While we rely on the skills of designers, working in 3D is conceptually hard. The kinds of authoring and debugging facilities Alice has would be quite useful in DART.

CONCLUSIONS

DART takes a significant step toward enabling designers to work with the exciting new medium of AR. By focusing on rapid prototyping and early experience testing, and creating an environment which allows designers to work effectively at (and away from) a physical site, we help designers get over the initial hurdle of creating a first working prototype of an embodied AR experience. We have demonstrated the importance of integrating a research toolkit with existing, powerful tools such as Director, and of addressing both technical (e.g., tracker integration) and practical problems (e.g., dealing with the annoyances of working in the physical world).

DART places a heavy emphasis on rapidly moving informal content from storyboards into a functioning AR prototype, and using this informal content to explore the design space. We believe that the difficulty of design exploration, not final content creation, is what has limited AR experience prototyping, it is this hole that DART has been designed to fill.

ACKNOWLEDGEMENTS

This work was supported by seed grants from the GVU Center at Georgia Tech and by NSF CAREER Grant 0347712. We acknowledge the contributions of our students, especially Emmanuel Moreno (who created our first AR experiences in Director) and the students in all our AR Design and independent research classes. We also thank UNC Chapel Hill for use of the VRPN library, and the UW HIT Lab for the ARToolkit.

REFERENCES

1. Abawi, Daniel F., Dörner, Ralf, Haller, Michael, Zauner, Jürgen. (2004) "Efficient Mixed Reality Application Development" In *1st European Conference on Visual Media Production (CVMP)*, London, England, March 15, pp. 289–294.
2. Billinghurst, M., Bowskill, J., Jessop, M., and Morphett, J. (1998) "A Wearable Spatial Conferencing Space", In *Proc. International Symposium on Wearable Computing (ISWC '98)*, pp. 76–93.
3. Conway, M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K., Deline, R., Durbin, J., Gossweiler, R., Koga, S., Long, C., Mallory, B., Miale, S., Monkaitis, K., Patten, J., Pierce, J., Shochet, J., Staack, D., Stearns, B., Stoakley, R., Sturgill, C., Viega, J., White, J., Williams, G., and Pausch, R. (2000) "Alice: Lessons Learned from Building a 3D System For Novices" In *Proc. CHI 2000*, pp. 486–493.
4. Feiner, S., MacIntyre, B., and Seligmann, D. (1993) "Knowledge-based augmented reality" *Communications of the ACM*, 36(7), July 1993, pp. 52–62.
5. Flintham, M., Anastasi, R., Benford, S., Hemmings, T., Crabtree, A., Greenhalgh, C., Rodden, T., Tandavanitj, N., Adams, M., and Row-Farr, J. (2003) "Where On-Line Meets On-the-Streets: Experiences with Mobile Mixed Reality Games" In *Proc. of CHI'03*, pp. 569–576.
6. Greenberg, S. and Fitchett, C. (2001) "Phidgets: Easy Development of Physical Interfaces through Physical Widgets." In *Proc. Symposium on User Interface Software and Technology (UIST '01)*, pp. 209–218.
7. Igarashi, T., Matsuoka, S., Tanaka, H. (1999) "Teddy: A Sketching Interface for 3D Freeform Design" In *ACM SIGGRAPH'99*, Los Angeles, pp.409–416.
8. Landay, James A., and Myers, Brad A. (2001) "Sketching Interfaces: Toward More Human Interface Design." *IEEE Computer*, vol. 34, no. 3, pp. 56–64.
9. MacIntyre, B., Bolter, J. D., Vaughn, J., Hannigan, B., Gandy, M., Moreno, E., Haas, M., Kang, S.-H., Krum, D., and Voida, S. (2003) "Three Angry Men: An Augmented-Reality Experiment in Point-of-View Drama" In *Proc. International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE'03)*, Darmstadt, Germany, pp. 230–236.
10. MacIntyre, B. and Feiner, S. (1998) "A Distributed 3D Graphics Library." In *Proc. ACM SIGGRAPH 98*, July 19–24, Orlando, Florida, pp. 361–370.
11. MacWilliams, A., Reicher, T., Klinker, G., and Bruegge, B. "Design Patterns for Augmented Reality Systems" In *Proc. International Workshop Exploring the Design and Engineering of Mixed Reality Systems - MIXER 2004*, Funchal, Madeira.
12. Moreno, E., MacIntyre, B., and Bolter, J.D. (2001) "Alice's Adventure's in New Media: An Exploration of Interactive Narratives in Augmented Reality." In *Conference on Communication of Art, Science and Technology (CAST'01),* Bonn, Germany, pp. 149-152.
13. Piekarski, W. and Thomas, B. H. (2003) "An Object-Oriented Software Architecture for 3D Mixed Reality Applications" In *Proc. International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp. 247–257.
14. Schell, J. and Shochet, J. (2001) "Designing Interactive Theme Park Rides: Lessons From Disney's Battle for the Buccaneer Gold" in *Gamasutra*, July 6, 2001.
15. Schmalstieg, D., Fuhrmann, A., Hesina, G., Szalavári, Z., Encarnação, L. M., Gervautz, M. and Purgathofer, W. (2002) "The Studierstube Augmented Reality Project" *PRESENCE–Teleoperators and Virtual Environments*, 11(1): 32–54.