The Evolution of a Framework for Mixed Reality Experiences

Charles E. Hughes, Christopher B. Stapleton, Matthew R. O'Connor
School of Electrical Engineering and Computer Science, School of Film and Digital Media
and the Institute for Simulation and Training at the
University of Central Florida, Media Convergence Laboratory (www.mcl.ucf.edu)

ABSTRACT

This chapter describes the evolution of a software system specifically designed to support the creation and delivery of Mixed Reality (MR) experiences. We first describe some of the attributes required of such a system. We then present a series of MR experiences that we have developed over the last four years, with companion sections on lessons learned and lessons applied. We conclude with several sample scripts that one might write to create experiences within the current version of this system. The authors' goals are to show the readers the unique challenges in developing an MR system for multimodal, multi-sensory experiences and to demonstrate how developing MR applications informs the evolution of such a framework.

KEYWORDS:

Mixed Reality, Virtual Reality, Augmented Reality, Augmented Virtuality, Distributed Systems, Interactive Technology, Scripting, Video See-Through Head Mounted Display, Software Agents, Object-Oriented

INTRODUCTION

Mixed Reality (MR) presents unique challenges in its requirement to seamlessly integrate interacting virtual objects, audio landscapes, visual presentations, haptic feedback and show-control devices with real world objects such as human participants, props and physical settings. In this chapter, we describe the evolution of our MR framework. The key connections between each section are the lessons we learned along the way while developing ever more complex and diverse applications of MR. We also emphasize the system's flexibility in its design (components can be distributed or aggregated as appropriate for a given scenario's needs); its provisions for all modes of MR (ranging from Physical Reality to Augmented Reality to Augmented Virtuality to pure Virtual Reality); its support for diverse modes of experiencing MR (ranging from video-see through head-mounted displays to vision domes to desktop systems, and from unaltered physical environments to theatrical sets to unidirectional retro-reflective "caves"); its openness (all components not developed in our lab are from the open source community); its modularity (its plug-in architecture can accommodate other network protocols, new physics engines, new user interfaces, new interaction devices, new authoring interfaces and new AI components); and its adaptability (we added the concept of story-based rendering in one evening). Our framework has been field-tested with installations for entertainment, free-choice learning, training and cognitive rehabilitation. Each such application has revealed strengths and exposed weaknesses that informed our evolving design and implementation, and influenced our program of basic research.

The central component of the MR framework is the MR Story Engine (SE), a container for software agents (actors): one for every user, virtual object and real object that interacts with other agents; one for communication with the underlying system; zero or more to maintain the story line; and zero or more to support abstractions. The software agents manage the semantics of the story in that their states and behaviors determine much of what a user sees, hears and feels. The SE allows for a wide range of functionality through its object-based scripting language. Language features include prototype-based object-oriented programming, real-time binding of variables (the names that reference variables can be created at run-time) and dynamic array sizes (that's the scheme for dynamic memory allocation). At run-time, agents in the SE can interact with interdependent engines (such as graphics and audio), and read/activate sensors and actuators (such as tracking and special effects devices). The SE provides an integrated physics engine, a pluggable interface for auxiliary physics engines and other autonomous behaviors, a protocol for graphical user interfaces, and both an abstraction and a network-based protocol for interaction devices. See Figure 1 for a visual depiction of the integration of some of these components. In this, the (1) video see-through HMD captures the real world scene. This captured imagery is then augmented based on (2) the user's orientation and position as sensed through tracking and (3) the states/behaviors of the agents as directed by the Story Engine and carried out by the Graphics, Audio and SFX (special effects) Engines. The user's experience is then provided through (A/B/C) visual and (E) auditory landscapes, and via (D) special effects.

To set our MR framework in context, we first discuss the Canon MR System and our needs to extend it. We then present examples of some experiences we have developed and how each influenced our system's evolution. The end result is a system that can support experiences as diverse as interactive story-telling, collaborative design, technology demonstrations, games, education, training and rehabilitation.

CANON'S MRPlatform

Our first MR system was built around Canon's work in Mixed Reality. The Canon team, under the direction of Hideyuki Tamura, had already developed several influential experiences, including AR^2 Hockey (Ohshima et al., 1998), AquaGuantlet (Tamura, 2001), Contact Water (Murakami et al., 2001) and others (Tamura et al., 2001). The evolution of these projects resulted in the creation of the MRPlatform API and library (Uchiyama et al., 2002). This is a Linux-based system that captures, analyzes, manipulates and delivers video, as seen through the Canon COASTAR video see-through HMD's cameras.

As revolutionary as Canon's system was, its API focused almost exclusively on the visual aspects of an MR experience and provided no scenario creation support beyond integration with OpenGL (http://opengl.org) and Open Inventor (http://oss.sgi.com/projects/inventor/). Our first extensions were to create Story, Audio and SFX Engines, each written in Java for platform portability. In this context, the Canon MRPlatform handled video capture and acquired data from the hardware tracker. Our Graphics Engine extended this with a set of C++ classes and services for integrating virtual and real content and displaying these composite scenes. To support engine distribution and communication, we developed a low latency network protocol through which the Story Engine could function as the center of control, sending requests to each of the other engines to trigger the visual, auditory and special effects events that contributed to an MR experience.

MR EXPERIENCES

We briefly describe four MR experiences: MR Time Portal, MR Sea Creatures, MR Kitchen and MR MOUT. (Hughes et al., 2005) The various versions of MR MOUT, developed through 6 iterations, most accurately reflect the evolution of our system. However, since that evolution both influenced and was influenced by the other applications, we describe its final iteration last. In all cases, we discuss lessons learned and lessons applied, using these to motivate the evolution of our software systems.

MR Time Portal

MR Time Portal, publicly shown at SIGGRAPH 2003, was the first widely seen experience we developed that involved complex 3D models with rich animations and a non-trivial story line. Its goal was to immerse participants within a story, with some people at the center of the action, and others at the periphery. Figure 2a is a scene from an animatic[1] we produced that helped us to test story elements while still in VR mode. Figure 2b shows the full MR with one person on the right wearing an HMD in order to be embedded in the experience and two people at a vision dome on the left observing the experience from the perspective of an unseen second participant. In essence, this is an MR version of a theme park experience employing those venues' notion of divers (the ones who get in the action), swimmers (those who influence the action) and waders (those who observe from afar). (Stapleton & Hughes, 2003; Stapleton & Hughes, 2005)

Our Evolving Story Engine

In 2003, our Story Engine was based on the concept of Java objects holding the states and primitive behaviors of actors, each having an associated finite state machine (Coppin, 2004) that controlled the manner in which these behaviors were invoked based on stimuli such as timed events, GUI inputs and interactions with other actors. Most actors reflected the virtual and active real objects of the MR world, but some existed to play the roles of story directors, encouraging players in directions deemed most supportive of the underlying story. For instance, MR Time Portal contained actors associated with a back-story movie; the portal through which threats to our world arose; various pieces of background scenery; each robotic threat; each friendly portal guard; a futuristic physical weapon; a ray tracing beam to make it easier to aim the gun; a number of virtual explosions; the lighting rig above the exhibit area; several abstract objects operating as story directors; and, of course, the real persons who were experiencing this world. Each of these actors had optional peers in our Graphics Engine, Audio Engine(s), and Special Effects Engine. The reason these are optional is that, at one extreme, abstract actors have no sensory peers; at the other extreme, robotic threats have visual representations, audio presentations that are synchronized in time and place with the visuals, and special effects when the robots hit the ground (a bass shaker vibrates the floor under the shooter); and in between are things like the lighting rig that has only a special effects peer.

An actor component, when added to the authoring system, had a set of core behaviors, based on its class. An Actor class sat at the top of this hierarchy, providing the most common default behaviors and abstract methods for required behaviors for which no defaults exist. A finite state machine, consisting of states and transitions, was the primary means of expressing an actor's behavior. Each

transition emanated from a state, had a set of trigger mechanisms (events) that enabled the transition, a set of actions that were started when the transition was selected, and a new state that was entered as a consequence of carrying out the transition. A state could have many transitions, some of which had overlapping conditions. If multiple transitions were simultaneously enabled, one was selected randomly. The probability of selecting a particular transition could be increased by repeating a single transition many times (the cost was just one object handle per additional copy). States and transitions could have associated listeners causing transitions to be enabled or conditions to be set for other actors.

Lessons Learned and Lessons Applied

In order to stay within the time constraints of completing Time Portal in the few months before SIGGRAPH 2003, we retained one part of our existing closed system, Canon's MRPlatform, and added another, Granny 3D (http://www.radgametools.com/gramain.htm), which provides capabilities that allowed us to smoothly transition between animations, e.g., from running to kneeling down. The problem with each of these is that the experiences created by them could not be distributed due to the proprietary nature of Canon's API and the licensing costs of Granny 3D.

By this time we had developed our own audio engine, MRAudio, and a soundscape authoring system, SoundDesigner. Both of these were based on OpenAL with EAX extensions (http://www.openal.org). This new audio suite allowed us to place sounds in 3D space using OpenAL's model of sources whose attached sound buffers are capable of being heard by a single listener based on that person's position and orientation relative to those of the sound source. While suitable for most dramatic effects, this version still did not meet our goals of total control of individual speakers. That feature was deferred to our next release of the audio suite.

This version of our system used two protocols for communication between the story and other engines. TCP/IP was used for communication to the Graphics Engine(s), and multi-cast UDP was used for communication to Audio and SFX Engine(s). We did this with the belief that missing a sound like a gun shot was acceptable, but missing a command such as starting an animation was not. Later tests showed that we got very little gain in performance out our use of UDP and that our primary network problem was congestion due to the manner in which we shared positional and orientation data..

Perhaps the biggest overarching lesson we learned from the development of Time Portal was the near impossibility of succeeding on such a complex project without having a well thought-out production pipeline in place (Hughes et al., 2005).

MR Sea Creatures Experience

MR Sea Creatures is described in detail elsewhere (Hughes et al., 2005). Here we just note that it takes place in the Orlando Science Center's DinoDigs exhibit, augmented with Cretaceous sea life that appear to swim around the pillars of the hall (Figure 3). (Hughes et al., 2005) The visitor is able to navigate a Rover through the ocean environment to explore the reptiles and plant life. The viewing window of the Rover is what the visitor sees in the Heads-Up Display. Thus, this application brings together physical reality, augmented reality and virtual reality in a single experience.

Tension is greatly enhanced by integrating the audio with the visual experience. Achieving this effect is partially technical – one needs to control the soundscape through a sophisticated delivery system, but the technical does nothing without the capture and/or synthesis of the "right" sounds. To capture turbulent underwater sounds, we made a custom-designed "XY" mount to position four hydrophones, enabling us to capture four discrete channels of audio for surround sound play back (Hughes, 2005). Ocean ambience was captured at New Smyrna Beach, Florida using a multi-channel mobile recording unit. The turbulence created by crashing waves made for a realism that reflects the violent seas of the Cretaceous period.

Lessons Learned and Lessons Applied

While the interface worked well with gamers and children in the middle school years (12-14), younger children (4-11) tended to consume the experience and adults were confused by the interface unless taught by their children (Hughes et al., 2005). This clearly demonstrated the need to support multiple interfaces that can be tailored to different audiences.

A big winner in this experience was the audio, reflecting lessons learned in MR Time Portal and various aspects of MR MOUT (to be discussed later). While Sea Creatures did not have as rigid positioning requirements for audio as some of our other applications, given that the experience was restricted to the dome, it was our first public use of our new Audio Engine, built using PortAudio (Bencina & Burk, 2001). One aspect that distinguishes it from other audio engines is its support for dynamic delivery based on user-specified speaker constraints. This feature is extremely useful for public access areas like science centers where speaker placement constraints may change from time to time, and rarely are reproducible in the laboratory in which the development first occurs.

The visual aspects of this experience tested our Graphics Engine that is now based on OpenSceneGraph (http://www.openscenegraph.org/), and the Character Animation Library (http://cal3d.sourceforge.net/), a skeletal-based 3d character animation library written in C++. The change to Cal3d removed the last vestige of dependence on proprietary software (Granny3D), while giving up none of the features, such as occlusion models with or without blue-screens, or the scalable performance we had previously achieved.

Sea Creatures was also successful in that it demonstrated the flexibility of our system, allowing us to swap out the HMD for an observer camera and dome; build an experience with multiple virtual views as well as a mixed reality view; run unattended for days; support interacting groups of participants, continuing the model started in Time Portal; embed the experience into its environment, masking the technology; have the appeal of a video game and the lasting memory of a theme park, while delivering relevant information and experiences; and be capable of being persistent (we could capture and replay people's interactions).

MR Kitchen Experience

The goal of the MR Kitchen was to demonstrate the use of MR in simulating a cognitively impaired person's home environment for purposes of helping that individual regain some portion of independence. More broadly, the goal was to experiment with the use of MR as a Human

Experience Modeler – an environment that can capture and replicate human experiences in some context. Here the experience was making breakfast and the context was the individual's home kitchen (Fidopiastis et al., 2005).

Experience Capture starts by recording the spatial, audio and visual aspects of an environment. This is done at the actual site being modeled (or a close approximation) so we can accurately reproduce the space and its multi-sensory signature. To accomplish this we employ a 3D laser scanner (Riegl LMS-Z420i), a light capture device (Point Grey Ladybug™ camera), and various means of acoustical capture (Holophone™ H2 Pro, stereo microphones on grids, transducers to pick up vibrations and sounds in micro environments, and even hydrophones for underwater soundscapes). Once captured, models of this real environment can be used to augment a real setting or to serve as a virtual setting to be augmented by real objects. This MR setting immerses a user within a multi-modal hybrid of real and virtual that is dynamically controlled and augmented with spatially registered visual, auditory and haptic cues.

For our MR Kitchen experiment, we went to the home of a person who had recently suffered traumatic brain injury due to an aneurism. Spending about two hours there, we "captured" his kitchen (see bottom right monitor of Figure 4 for an image of him in his home kitchen). This capture included a point cloud, textures, and the lighting signature of the kitchen and its surrounds (audio was not used for the experiment). We then built parts of the real kitchen out of plywood to match the same dimensions and location of critical objects (pantry, silverware drawers, etc.) We purchased a refrigerator, cupboard doors, a coffee maker and toaster oven, and borrowed common items (cups, utensils and favorite cereal). Figure 4 shows two participants in this kitchen. The screen on the left shows the view from the man on the right. Notice that the real objects are present; however the textures of the counter and doors are the same as in the subject's home.

All aspects of the subject's movement and his interaction with objects and the human therapist are captured by our system as seen in the center monitor of Figure 4. This capture includes a detailed map of the subject's movement and head orientation, allowing for analysis and replay. Additionally, cameras can be positioned to capture any number of observer viewpoints of his activities and those of his therapist. Replaying the experience allows viewing events from multiple perspectives and with appropriate augmentation (e.g., data on the user's response times and measured stress levels).

Lessons Learned and Lessons Applied

The recorded the activities of our participant showed dramatic improvement in his accessing needed (bowl, spoon, milk and cereal) over a five-session training period. This provides anecdotal evidence that MR is an effective means for helping in the rehabilitation of an adult who has suffered brain injury. Transfer of learning from the MR environment to his home environment was evidenced in decreased time spent on task, decreased number of location errors, and decreased wandering behavior. The after action review, supported by experience capture/replay from multiple points of view, including that of the participant, was an essential contribution.

MR MOUT Experiences

The MR MOUT (Military Operations in Urban Terrain) test bed is a training simulation that recreates urban façades to represent a 360 degree mini MOUT site. Tracking employs the Intersense IS-900 acoustical/inertial hybrid system. The tracked area contains virtual people (friends, foes and neutrals), real props (crates, doors, and a swinging gate), a realistic tracked rifle, real lights and building façades. Standing inside the mini MOUT creates the sense of reality faced by a dismounted soldier who is open to attack on all sides and from high up.

Using a combination of blue screen technology and occlusion models, the real and virtual elements are layered and blended into a rich visual environment. The trainee has the ability to move around the courtyard and hide behind objects with real and virtual players popping out from portals to engage in close-combat battle. The most effective and powerful result of this mixed reality training is the fact that the virtual characters can occupy the same complex terrain as the trainees. The trainees can literally play hide and seek with virtual foes, thereby leveraging the compelling nature of passive haptics.

Figure 5 shows the mini MOUT from the observer station. In the middle, to the right of the observer, you can see the participant with HMD and rifle. That person's view is shown on the screen mostly blocked by the observer; the other three views are from an observer camera (middle right) and two virtual characters (lower right and top center). Notice the crates in the view on the middle right side. The models that match these physical assets are rendered invisibly, providing appropriate occlusion (they clip the rendered images of characters that they would partially or totally occlude in the real world). Special effects complete the creation of a realistic combat scenario where the real world around the trainee feels physically responsive. This is done using the SFX Engine to control lights, smoke from explosions, and other types of on/off or modulated actions. The system can react to the trainee based on position, orientation or actions performed with a gun that is tracked and whose trigger and reload mechanism are sensed. For example, the lights on the buildings can be shot out (we use a simple ray casting auxiliary physics engine that returns a list sorted by distance of all intersected objects), resulting in audio feedback (the gunshot and shattered glass sounds) and physical world visual changes (the real lights go out).

With all the compelling visual and haptic effects, one's hearing and training can provide a competitive edge, due to a heightened acoustical situational awareness (Hughes et al., 2004). You can't see or feel through walls, around corners or behind your head. However, your ears can perceive activity where you cannot see it. In urban combat where a response to a threat is measured in seconds, realistic audio representation is vital to creating a combat simulation and to training soldiers in basic tactics. Standard 3D audio with earphones shuts out critical real-world sounds, such as a companion's voice or a radio call. The typical surround audio is still two-dimensional (x and z axis) with audio assets designed for a desktop video game that tend to flatten the acoustical capture. Our system allows audio to be synchronized temporally and spatially, leading to an immersive experience.

Lessons Learned and Lessons Applied

Most of the lessons learned during the various versions of MR MOUT have already been discussed in previous sections. One requirement very specific to MOUT was its demand for long-term

hands-off deployment. While the science center had this characteristic, we were able to enter the center at night to do fine-tuning, and the experiment was for only three weeks. In contrast, MOUT is inaccessible for longer periods of time, is never available at night, and has far more restricted access since entrance to and operation in the Army facility requires clearances that we didn't have. MOUT was also the place where we first needed and developed 360 degree audio whose position and orientation cannot be delivered to just one listener, but rather must be precisely associated with dynamically changing 3d positions (within the constraints of speaker placements).

THE CURRENT MR FRAMEWORK

The current incarnation of our framework utilizes an XML scripting language, based on the concepts of interacting agents, behaviors, guards, and state information. This separates the script-writers from the internals of the engine, while providing them a meaningful and effective context in which to encode simple, direct behaviors (O'Connor & Hughes, 2005). The reorganization of the system prompted the development of other supporting engines, dubbed Auxiliary Physics Engines, or APEs. These engines are responsible for tasks such as path-finding and ray-casting, since our revised architecture attempts to make distinct and clear the tasks of each engine.

The philosophy of a distributed system was key to the construction of this framework. The Story Engine is the hub, providing script-writers access to any presentation requirements they need. For complex cases, our XML-based script language allows one to escape into a special sub-language, dubbed the Advanced Scripting Language, or ASL. The ASL provides the ability to code behaviors using C-style programming constructs. Such constructs include loops, conditionals, basic arithmetic, and assignment operations.

The script defines a set of *agents*, each of which generally embodies some character the user may interact with (directly or indirectly). Agents are defined in terms of behaviors, which include actions, triggers, and reflexes, and a set of state variables that define an agent's current state. Each behavior can perform several tasks when called, such as state modification and the transmission of commands to the presentation engines. Thus, agents are the fundamental building-blocks of the system. The ability for agents to communicate with each another allows for a "world-direct" representation to be built: you define a set of agents in terms of how you want them to act around each other, rather than such actions being a side-effect of a more program-like structure.

The Graphics and Audio Engines understand the same basic set of commands. This allows the script-writer to easily generate worlds that offer visual and audio stimulation. Each engine also has a set of commands unique to its particular functionality, e.g., audio clips can be looped and visual models can have associated animations.

The SFX Engine utilizes the DMX protocol, but control over it originates from the Story Engine through a series of commands, most of which are defined by loadable "DMX scripts." These scripts are direct control specifications that offer a set of basic functions (typically setting a device to a value between 0 and 255 meaning off to fully on and anything in between). These primitives are hooked together to form complex DMX events.

In our older versions of the system, agent information, such as position and orientation, was managed by the Graphics Engine. This required the Story Engine to request regular updates, thus causing network congestion when the number of agents was high. The current incarnation of the system does away with this, and now all physics simulation is performed by the Story Engine. The data is transmitted as a binary stream, encapsulated in a cross-platform and cross-language format. The data stream is denoted the "control stream," as it controls the position, orientation, velocities and accelerations of agents. A given control stream is broken up into numbered channels, one channel for each agent (channel numbers are automatically assigned to agents and are accessible through the reserved state variable name *channel*). This enables us to transmit only a subset of the data, usually only that which has changed since the last transmission. The system scales remarkably well.

Many of the distributed capabilities not only involve the major engines, but also a set of utility servers. One type of utility server, the Auxiliary Physics Engine, was referenced earlier. Two APEs were developed for our projects: one to control path-finding on a walk-mesh, and another to manage ray-casting in a complex 3D universe. These engines plug in at runtime, and simply serve the requests of agents.

Another utility server is the Sensor Server. This basically abstracts data from position and orientation sensors into data streams, which are then transmitted across a network to interested clients. This allows any number of agents to utilize the data. The data stream is transmitted via TCP/IP for reliability purposes. The data format follows that of the Story Engine's control stream data. Thus, to a Graphics or Audio Engine, it is immaterial where control data comes from; a given agent's control may be governed by the user's own movements, or that of a simulated entity's. The Sensor Server also enables us to record user movement, a vital piece of information for after-action review (a military training term, but equally important for rehabilitation applications such as MR Kitchen) and cognitive experimentation.

The ability to define a set of behaviors to be reused in several scripts came to life in the "script component" architecture. This architecture allows "component" files to be written by the script-writer, and then included in any number of scripts. Behaviors or entire agents can be scripted, and consequently included, into the main script. This also means that difficult-to-code behaviors and algorithms can be written once and used repeatedly, without having to perform copy-and-paste operations or rename a vast number of states and agents. The Story Engine allows object-oriented capabilities such as prototype-based inheritance and delegation to make coding agents reasonably straight-forward and simple.

A final and rather recent innovation to the architecture was that of a remote system interface. Originally designed as an interface to allow remote (over-the-network) access to agent state information for display on a graphical user interface, the Remote GUI protocol also provides a way to transmit information back to the Story Engine. It is, in effect, a back-door into the virtual world controlled by a given script, whereby agent command and control can be affected by a purely remote, alien program. We recently took advantage of this capability to link our system to DISAF (Dismounted Infantry Semi-Automated Forces), an AI system that provides behaviors used in many Distributed Interactive Systems (DIS) applications.

Used for its original purpose, the Remote GUI protocol and program architecture allows graphical interfaces to be defined by a simple XML file. The file specifies graphical components to be used, as well as options for each component that link it to agents and states in the script. An example of this would be to display the number of times a particular agent was encountered by the user: a simple state variable in the agent itself would keep track, and changes to that information would be retrieved and displayed by the Remote GUI. This approach is amenable to a drag-and-drop approach for creating such GUIs, something we will do in the next release of the software.

SAMPLE SCRIPTS

Figure 6 is a snippet of script associated with a rifle that is tracked separately from the user's position and orientation. This shows the "init" script that is called when the agent is initially loaded by the SE and two actions, "fire" and "reload." The "fire" message is sent by the SYSTEM object when the trigger is pulled (SYSTEM can provide callbacks for all external events); the "reload" is sent when the magazine reload mechanism is pulled back and released. In this simple case, most commands are messages to the Graphics or Audio Engines or a device called the haptic vest. These are denoted by the tags <gfxcommand>, <audcommand> and <devcommand device="HapticVest" channel=…>, respectively.

The rifle agent maintains state for the physical rifle. This includes ammunition remaining and the next buzzer on the haptic vest to be enabled, plus position and orientation, state variables that are inherited by any agent that is associated with type "model." The actual rendered model here is an occlusion model for an M4, kept in the props folder for the MOUT3 (Military Operations in Urban Terrain Version 3) experience; the audio is in the corresponding audio folder. By occlusion model, we mean that, when the rifle is shown, it is actually rendered invisibly, but its presence results in occlusion (partial or total) of virtual objects that it would occlude were it rendered visibly. The effect is that the real rifle appears to visually occlude virtual objects whose line of sight is blocked from the user. This is extremely important in retaining the suspension of disbelief needed to accept that the real and virtual share the same space.

The most complex statements in "init" are the ones that tie the rifle model's position and orientation to the actual rifle. This is achieved by "rifle set control 2@SensorServer:4095" causing the rifle's control information to come from logical stream 2 at TCP/IP port 4095 on a node named SensorServer. The node name can be anything that resolves to an IP address, including "localhost" if the sensor server is on the same node as the SE. Typically this is done symbolically, allowing changes to be done via configuration files or even through a GUI. The first stream on the sensor server port is dedicated to the first user and the second to that person's rifle. Other streams can be associated with other sensors including those for additional users and additional interaction devices.

You will note that the "show" commands for the rifle differ between the Graphics and Audio Engines. The "world show" and "world set mode occlude for agent rifle" commands for graphics place the rifle in the collection of objects that will be rendered invisibly, thus creating the desired occlusion effect. The "SurroundChannel show rifle" command tells the Audio Engine that it should use the speakers in the SurroundChannel collection to accurately deliver the sounds associated with the rifle to the current position of the rifle, as specified by the Sensor Server. Where appropriate sounds can be associated with fixed channels, e.g., the sounds delivered

through a headset would be played on the channels connected to that head set, not on the speakers set up in the environment.

The first action (behavior) of this rifle is to "fire." As noted above, this communiqué (message) is sent when the trigger is depressed. This behavior has two cases. The first is enabled when its one guard, ammo>0, is true. It results in ammo being decremented by one and a sound clip, rifleshot, being played as if produced at the position of the gun. The second is enabled when its guards, ammo=0 and buzzer<8, are true. This means the user tried to shoot with no ammunition remaining. When this first occurs we set an actuator on the haptic vest to give the person a slight buzzing sensation in the front of the left shoulder. If the user tries to shoot again without reloading, the next actuator is turned on, giving a buzzing sensation on the front of the right shoulder. All in all, we have actuators at eight points, four in the front and four in the back. Once all are turned on, pulling the trigger does nothing else because the case is now disabled (buzzer=8).

The only way for a user to stop the haptic vest from buzzing is to reload the ammunition magazine. When this is done, ammo gets reset to 30, buzzer gets reset to 0, a sound is played to simulate the loading of the magazine, and all the channels of the vest are deactivated (set to 0). The deactivation could be done by a series of eight results, but ASL's support for iteration makes this easier. The tags <asl<![CDATA[escape to ASL and C syntax, allowing us to write a typical "for loop." The only oddity here is the surrounding of buzzer, {@buzzer}. The reason for this is that buzzer by itself would be interpreted as the string "buzzer" not the value stored in buzzer, whereas @buzzer is the value. The surrounding braces demarcate the start and end of such an evaluation, allowing new strings to be dynamically formed, such as with buzz{@buzzer}on, which would become the name buzz3on if buzzer=3. This ability to dynamically create names can lead to very compact but potentially cryptic code. It, as well as many other language features, are constantly being evaluated. Fortunately, our remote interface allows us to attach behaviors created in any language so long as the new API implements our communication protocol.

Demonstrating all the features of the scripting language would consume another chapter, so we direct the reader to http://mcl.ucf.edu/software/MRSS/docs/StoryEngine/ at which detailed descriptions can be found. For now we just note that our system supports blue-screening just by the command

<gfxcommand hold="n">world set mask on for agent {@SELF}</gfxcommand>

to turn on blue screening (the character's rendering appears only if in front of a blue screen background), and

<gfxcommand hold="n">world set mask off for agent {@SELF}</gfxcommand>

to turn it off. As this is often used in conjunction with a character entering and exiting a building (inside, use blue screen; outside, do not), regions within the scene can be used to trigger messages to agents telling them when they make such transitions. As an example, the scene in Figure 7 has one region. If any object (denoted by being the SOURCE field of the argument ARGS) enters this region {-1000 to 1000 east to west (one meter in each direction from the center)}, {3000 to 9999 north to south (basically anything three meters or beyond in the north end))} and {0 to 4000 in the vertical (ground to four meters high)}, then the object will be blue-screened. Once it exits this region it will stop using the blue screen and thus be rendered unclipped, except where occluded. Note that regions can exist for reasons other than blue-screening, e.g., to cause a character to fall if it enters a region that represents a hole in the ground.

CONCLUSIONS AND FUTURE RESEARCH

This chapter describes the evolution of one specific system for authoring and delivering Mixed Reality experiences. We make no specific claims about its comparative benefits over other systems such as AMIRE (Traskback, 2004), MX Toolkit (Dias et al., 2003), Tinmith-evo5 (Piekarski et al., 2003) and DELTA3D (http://www.delta3d.org). Rather, our goal is to note the challenges we faced creating complex MR experiences and, within this context, to describe our means of addressing these issues.

As in any project that is coping with an evolving technology, we must sometimes provide solutions using existing and new technologies, e.g., solving clipping problems with blue screens and then employing unidirectional retro-reflective material in contexts that require the dramatic effects of changing real light. Other times we need to develop new scientific results, especially in the algorithmic area as in addressing realistic illumination and associated shading and shadowing properties in interactive time (Konttinen et al., 2005). Yet other times we must create new artistic conventions to deal with issues not easy solved by technology or science, e.g., taking advantage of people's expectations in audio landscapes (Hughes et al., 2004).

We believe that the most important properties of the framework we evolved are its use of open software, its protocols for delivering a scalable distributed solution and its flexible plug-in architecture. In general, flexibility in all aspects of the system has been the key to our success and is helping us to move forward with new capabilities, such as a bidding system for story-based rendering.

In its present form, our framework still requires scripts to be written or at least reused to create a new experience. Our goal (dream) is to be able to use our experience capture capabilities to evolve the behaviors of virtual characters in accordance with actions performed by human participants, as well as those of other successful virtual characters. For instance, in a training environment, the actions of an expert at room clearing could be used to train virtual SWAT team members by example. In a rehabilitation setting, the actions of a patient could be used as a model for those of a virtual patient that is, in turn, used to train a student therapist in the same context. Of course, this is a rather lofty goal, and just making authoring more intuitive, even with drag-and-drop, would help.

The MR framework described here is a system that is intended to generate, deploy, capture, analyze and synthesize an interactive story. Whether these stories are designed to train, teach, sell or entertain is immaterial. The point is that we drive an MR experience by generating a world within, on top, beneath, and around the real world and real senses that we live in. Our goals for this framework and for Mixed Reality in general are bounded only by our temporal imagination. Tomorrow, we will conceive of new applications of MR, leading to new requirements that continue to guide the evolution of our system and place new demands on our creativity.

ACKNOWLEDGEMENTS

The research reported here is in participation with the Research in Augmented and Virtual Environments (RAVES) supported by the Naval Research Laboratory (NRL) VR LAB. The MR MOUT effort is funded by the U.S. Army's Science and Technology Objective (STO) Embedded Training for Dismounted Soldier (ETDS) at the Research, Development and Engineering

Command (RDECOM). Special thanks are due to the Mixed Reality Laboratory, Canon Inc., for their generous support and technical assistance. Major contributions were made to this effort by artists Scott Malo, Shane Taber and Theo Quarles, artist and script writer Nathan Selikoff, audio designer/engineer Darin Hughes, experience designer Eileen Smith, and computer scientists Nick Beato and Scott Vogelpohl.

REFERENCES

Bencina, R. & Burk, P. (2001). PortAudio – an Open Source Cross Platform Audio API. *Proceedings of International Computer Music Conference (ICMC 2001)*, Havana, Cuba, September 18-20.

Coppin, B. (2004). *Artificial Intelligence Illuminated*. Sudbury, MA: Jones and Bartlett Publishers.

Cruz-Neira, C., Sandin, D.J., DeFanti, T.A., Kenyon, R., & Hart, J.C. (1992). The CAVE, Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM, 35* (6), 64-72.

Dias, J.M.S., Monteiro, L., Santos, P., Silvestre, R. & Bastos, R. (2003). Developing and authoring mixed reality with MX toolkit. *IEEE International Augmented Reality Toolkit Workshop*, October 7, 18-26.

Fidopiastis, C. M., Stapleton, C. B., Whiteside, J. D., Hughes, C. E., Fiore, S. M., Martin, G. A., Rolland J. P., & Smith, E. M. (2005). Human Experience Modeler: Context Driven Cognitive Retraining and Narrative Threads. *Proceedings of 4th International Workshop on Virtual Rehabilitation (IWVR2005)*, Catalina Island, CA, September 19-21.

Hughes, C. E., Stapleton, C. B., Hughes, D. E. & Smith E. (2005). Mixed Reality in Education, Entertainment and Training: An Interdisciplinary Approach. *IEEE Computer Graphics and Applications, 26* (6), 24-30.

Hughes, C. E. & Stapleton, C. B. (2005). The Shared Imagination: Creative Collaboration in Augmented Virtuality. *Proceedings of Human Computer Interaction International 2005 (HCII2005)*, Las Vegas, NV, July 22-27.

Hughes, D. E. (2005). Defining an Audio Pipeline for Mixed Reality. *Proceedings of Human Computer Interaction International 2005 (HCII2005)*, Las Vegas, NV, July 22-27.

Hughes, D. E., Thropp, J., Holmquist J. & Moshell, J. M. (2004). Spatial Perception and Expectation: Factors in Acoustical Awareness for MOUT Training. *Proceedings of 24th Army Science Conference (ASC 2004)*, Orlando, FL, November 29-December 2.

Konttinen, J., Hughes C. E. & Pattanaik, S. N. (2005). The Future of Mixed Reality: Issues in Illumination and Shadows. *Journal of Defense Modeling and Simulation, 2* (1), 51-59.

Murakami, T., Morita, K., Okuno, Y., Shimoyama, T. & Yonezawa, H. (2001). Contact Water. *Proceedings of International Symposium on Mixed Reality (ISMR 2001)*, Yokohama, Japan, March 14-15, 215.

O'Connor M. & Hughes, C. E. (2005). Authoring and Delivering Mixed Reality Experiences. *Proceedings of 2005 International Conference on Human-Computer Interface Advances in Modeling and Simulation (SIMCHI'05)*, New Orleans, January 23-27, 33-39.

Ohshima, T., Satoh, K., Yamamoto, H. & Tamura, H. (1998). AR2 Hockey: A Case Study of Collaborative Augmented Reality. *Proceedings of the Virtual Reality Annual international Symposium (VRAIS 1998)*, Washington, DC, March 14-18, 268.

Piekarski, W. & Thomas, B.H. (2003). An object-oriented software architecture for 3D mixed reality applications. *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, October 7-10, 247-256.

Stapleton, C. B. & Hughes, C. E. (2006). Believing is Seeing. *IEEE Computer Graphics and Applications, 27* (1), 88-93.

Stapleton, C. B. & Hughes, C. E. (2005). Mixed Reality and Experiential Movie Trailers: Combining Emotions and Immersion to Innovate Entertainment Marketing. *Proceedings of 2005 International Conference on Human-Computer Interface Advances in Modeling and Simulation (SIMCHI'05)*, New Orleans, January 23-27, 2005, 40-48.

Stapleton, C. B. & Hughes, C. E. (2003). Interactive Imagination: Tapping the Emotions through Interactive Story for Compelling Simulations. *IEEE Computer Graphics and Applications, 24* (5), 11-15.

Tamura, H. (2001). Overview and Final Results of the MR Project. *Proceedings of International Symposium on Mixed Reality (ISMR 2001)*, Yokohama, Japan, March 14-15, 97-104.

Tamura, H., Yamamoto, H., and Katayama, A. (2001). Mixed Reality: Future Dreams Seen at the Border between Real and Virtual Worlds. *IEEE Computer Graphics and Applications, 21* (6), 64-70.

Traskback, M. (2004). Toward a Usable Mixed Reality Authoring Tool. *2004 IEEE Symposium on Visual Languages and Human Centric Computing*, September 26-29, 160-162.

Uchiyama, S., Takemoto, K., Satoh, K., Yamamoto, H. and Tamura, H. (2002). MR Platform: A Basic Body on Which Mixed Reality Applications Are Built. *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2002)*, Darmstadt, Germany, Sept. 30-Oct. 1, 2002, 246-256.

USITT. (1990). *DMX512/1990 & AMX 192 Standards*. Syracuse, NY: United States Institute for Theatre Technology, Inc.

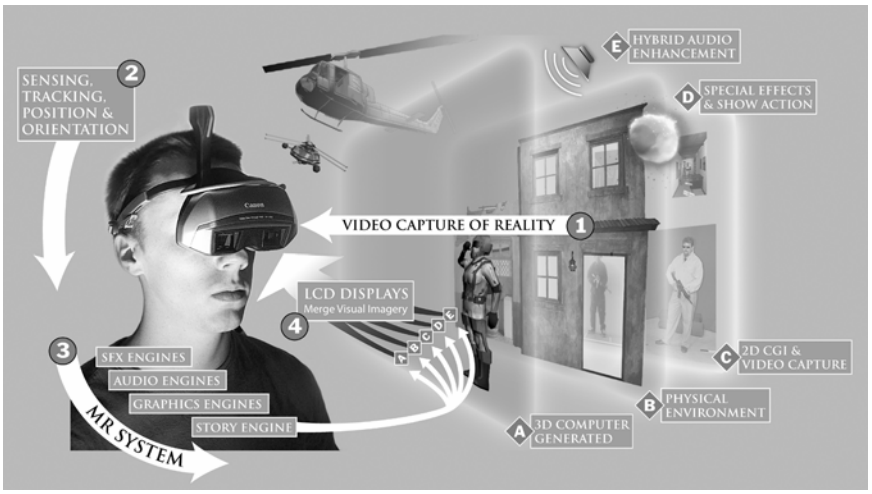
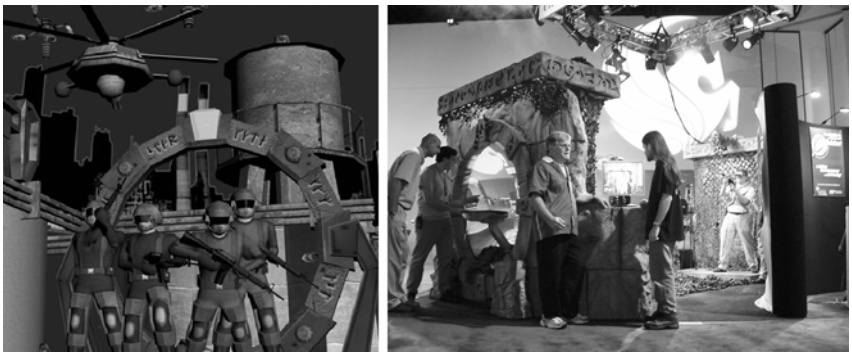


Figure 1. The integration of engines and devices in the MR framework *(©2006 UCF)*



(a) (b)

Figure 2. MR Time Portal: Experiential Movie Trailer – (a) Animatic; (b) Mixed *(©2006 UCF)*

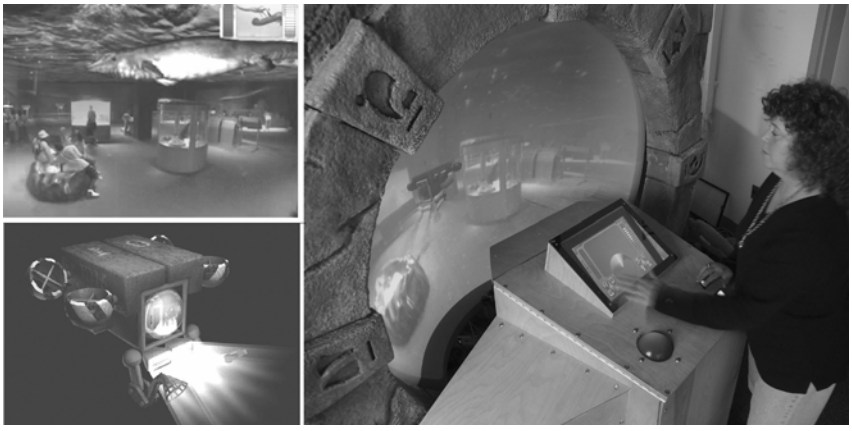


Figure 3. MR Sea Creatures: Free Choice Learning at Orlando Science Center *(©2006 UCF)*

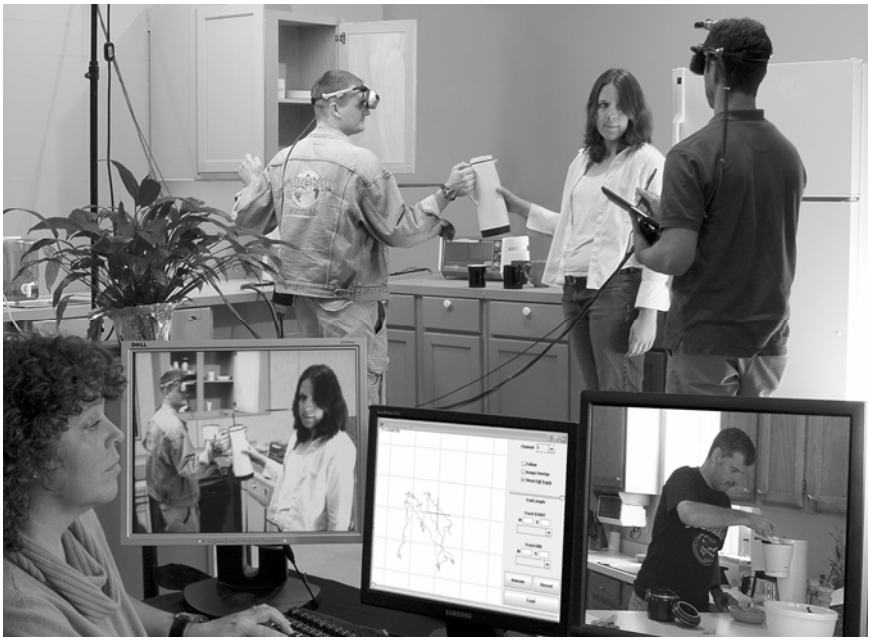


Figure 4. MR Kitchen: Cognitive Rehabilitation – Demonstrates real and mixed views, captured movement data for one participant, and subject's home kitchen *(©2006 UCF)*

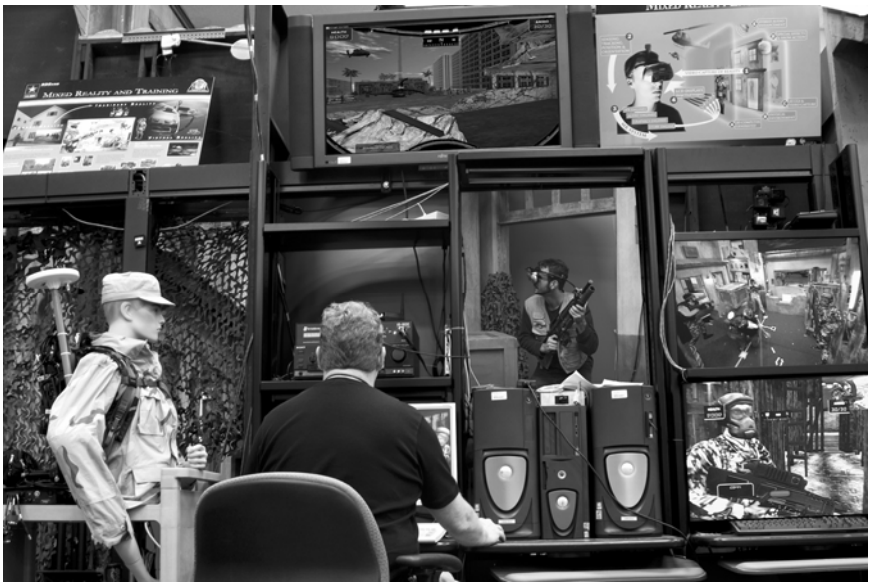


Figure 5. MR MOUT: Military Training – Demonstrates observer views (mixed and virtual) with a direct view of real world containing MR participant *(©2006 UCF)*

```
<agent name="rifle" type="model">
  <init>
    <gfxcommand>system make rifle as model</gfxcommand>
    <gfxcommand>rifle set model MOUT3_props.M4</gfxcommand>
    <gfxcommand>rifle set control 2@SensorServer:4095</gfxcommand>
    <gfxcommand>world add agent rifle </gfxcommand>
    <gfxcommand>world show rifle </gfxcommand>
    <gfxcommand>world set mode occlude for agent rifle </gfxcommand>
    <audcommand>system make rifle as model</audcommand>
    <audcommand>rifle set control 2@SensorServer:4095</audcommand>
    <audcommand>SurroundChannel show rifle </audcommand>
  </init>
  <stateset>
    <int name="ammo">30</int>
    <int name="buzzer">0</time>
  </stateset>
  <action name="fire">
    <case name="haveAmmo">
      <guard state="ammo"> <gt>0</gt> </guard>
      <result state="ammo"> <subt>1</subt> </result>
      <audcommand>gun play MOUT3_audio.rifleshot</audcommand>
    </case>
    <case name="noAmmo">
      <guard state="ammo"> <eq>0</eq> </guard>
      <guard state="buzzer"> <lt>8</lt> </guard>
      <result state="buzzer"> <add>1</add> </result>
      <devcommand device="HapticVest" channel={@buzzer}>255</devcommand>
    </case>
  </action>
  <action name="reload">
    <case name="always">
      <result state="ammo"> <set>30</set> </result>
      <result state="buzzer"> <set>0</set> </result>
      <audcommand>rifle play MOUT3_audio.reload</audcommand>
      <asl>
        <![CDATA[
          for(i = 0; i < 8; i += 1)
            <devcommand device="HapticVest" channel="{@i}">0</devcommand>
        ]]>
      </asl>
    </case>
  </action>
</agent>
```

Figure 6. Trivial rifle agent

```
<scene>
  <region name="OfficeBuilding" c1="-1000 3000 0" c2="1000 4000 4000">
    <action name="ENTERED">
      <case>
        <communique signal="maskOn" target="{@ARGS.SOURCE}"/>
      </case>
    </action>
    <action name="EXITED">
      <case>
        <communique signal="maskOff" target="{@ARGS.SOURCE}"/>
      </case>
    </action>
  </region>
</scene>
```

Figure 7. Regions and blue-screening

[1] An animatic is a simple visual rendering of the story from a single point-of-view. Its purpose is to communicate the vision of the creative team. This allows the art director, audio producer and lead programmer to effectively exchange ideas and determine each team's focus.